LISIANE MAES VOLPI

UMA ESTRATÉGIA DE TESTE DE SOFTWARE PARA AMBIENTE CLIENTE-SERVIDOR

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Curso de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora:
Prof.ª Dr.ª Silvia Regina Vergilio

CURITIBA
2001

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática da aluna ***Lisiane Maes Volpi***, avaliamos o trabalho intitulado ***"Uma Estratégia de Teste de Software para Ambiente Cliente Servidor"***, cuja defesa foi realizada no dia 31 de agosto de 2001, às dez horas, no anfiteatro B do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação da candidata.

Curitiba, 31 de Agosto de 2001.

Profª. Dra. Silvia Regina Vergilio
DINF/UFPR - Orientadora

Profª. Dra. Eliane Martins
IC/UNICAMP

Prof. Dr. Martin Alejandro Musicante
DINF/UFPR

AGRADECIMENTOS

Agradeço a Deus acima de tudo que permitiu transformar este sonho em realidade e tem sido comigo em toda a caminhada até aqui e é digno de toda a glória.

Agradeço ao meu amado esposo Harrald por toda a sua compreensão, o seu carinho e apoio que foram fundamentais em todo este trabalho.

Agradeço meus pais Leni e Volpi, que com tanto esforço lutaram para que eu pudesse ter uma profissão e pudesse caminhar com minhas próprias pernas e também pela compreensão da ausência durante este período. Aos demais entes queridos que compreenderam a ausência num longo período: meus queridos sogros Oli e Arno e também meu irmão e cunhada Xande e Selma e minha sobrinha Maria Luisa e meus cunhados Claus, Dorian, Nika e Ingo e sobrinhos Martina, Rafael e Natália.

Agradeço a CELEPAR que me liberou meio expediente por 22 meses, pela compreensão e apoio dos meus colegas, principalmente os colegas do setor em que trabalho no momento na GPT. Quero ainda fazer um agradecimento especial para Ozir, Cristina, Vidal, Sara e Edna que de alguma forma contribuíram com indicações, material, orientações e revisão do trabalho. Quero externar minha gratidão ao meu ex-chefe Dante que foi uma peça chave para a realização deste e também a uma pessoa que me encorajou muito, a Xana.

Quero agradecer meu grupo de discipulado (Silvio, Elaine, Eloá, Levi, Neiva, Renato, Claudia, Marcelo, Cristina, João, Ana, Jailson, Fábia) e também a vários amigos Wilson, Nelsi, Marco, Rosana, Ubiratan, Silvana, Pst. Josmar, Magda, Pst. Silas, Carmem, Silmara e Izabel, pelo apoio e orações nos momentos em que este sonho parecia um pesadelo.

Quero agradecer ao professor Luiz Callefe pelas orientações nas estatísticas do trabalho empírico. E em especial a minha orientadora professora Silvia que com tanta paciência executou o seu papel com louvor e foi a principal responsável pela concretização e qualidade desse sonho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AD - Administração de Dados

API – *Aplication Programming Interface*

ASP – *Active Server Page*

CMM – *Capability Maturity Model*

DLL – *Dynamic Link Library*

DSN – *Data Source Name*

GUI - *Graphical User Interface*

HTML – *HyperText Markup Language*

IEEE - *Institute of Electrical and Electronics Engineers*

ISO – *International Standard Organization*

KPA – *Key Practice Areas*

MDAC – *Microsoft Data Access Component*

MTS – *Microsoft Transaction Server*

ODBC – *Open DataBase Conectivity*

OLTP – *OnLine Transaction Process*

OOUI - *Object Oriented User Interface*

RUP - *Rational Unified Process*

SCM - Software *Configuration Management*

SDLC – Software *Development Life Cyclo*

SEI – Software *Engineering Institute*

SGBD - (DBMS–*DataBase Management System)* Sistema gerenciador de Banco de Dados

SPICE – Software *Process Improvement and Capability dEtermination*, atualmente, ISO/IEC TR 15504

SQL – *Structured Query Language*

STD – *Standard*

UML - *Unified Modeling Language*

UP - *Unified Process*

RESUMO

O teste é uma atividade fundamental para garantir a qualidade do software e tem sido uma meta nas empresas de desenvolvimento de sistemas. Nessas empresas, o uso da tecnologia cliente-servidor também tem se intensificado nos últimos anos.

Entretanto, este tipo de arquitetura cliente-servidor dificulta a atividade de teste por dividir o software em camadas. Muitos autores têm enfatizado a importância de testar o software para o ambiente cliente-servidor de uma maneira diferente.

Este trabalho propõe uma estratégia de teste denominada ETACS, **E**stratégia de **T**este de Software para **A**mbiente **C**liente-**S**ervidor. A ETACS é resultado de uma pesquisa realizada em uma empresa e de estudos de diversos trabalhos na literatura. A ETACS reúne alguns dos pontos positivos dos trabalhos estudados e procura resolver os problemas encontrados na pesquisa.

A ETACS é completa no sentido de orientar todas as etapas da atividade de teste, fornecendo roteiros que dão ênfase nas diferentes camadas do ambiente cliente-servidor, não vinculado a uma ferramenta. O trabalho também apresenta resultados de um estudo de caso, no qual a ETACS foi utilizada e avaliada.

ABSTRACT

Testing is a fundamental activity to assure the quality of a software and has been the goal of most software organizations. In organizations, the use of the client-server technology has been growing in the last years.

However, this kind of architecture makes the testing activity more dificulty, due to its different tiers. Many authors have pointed out the importance of testing client-server software in a different way.

This work proposes a specific client-server strategy, named ETACS – Software Testing Strategy for client-server environment. ETACS is a result of an empirical study done in a company and of studies of several works from avaiiable literature. ETACS considers some positive aspects of the studied works and intends to solve the main problems found in the empirical study.

ETACS is complete in the sense of considering all the testing phases. It gives some guidelines focusing the different tiers of the client-server software and it's not bound to any specific tool. The work also presents results from a case study applying ETACS.

1 INTRODUÇÃO

1.1 CONTEXTO

Teste de software é uma atividade crítica para garantia de qualidade de software. O processo de desenvolvimento de software envolve uma série de atividades onde a possibilidade de ocorrência de um erro é muito grande ou a interpretação incorreta da informação devido a falhas na comunicação humana é muito comum. Estes erros podem acontecer ao longo de todo o desenvolvimento, desde a concepção do software até a construção do mesmo. A atividade de teste propõe-se a auxiliar na descoberta destes erros, o quanto antes, para que o custo de correção dos mesmos seja o menor possível.

A informática, sendo uma empresa, departamento, setor ou área tem buscado uma melhor qualidade em seus produtos e serviços para melhor atender às necessidades de seus clientes e se manter em um mercado altamente competitivo [67]. Existe uma série de certificações e padronizações para a área de desenvolvimento de software. Nesse contexto, o modelo SEI/CMM (Software *Engineering Institute/Capability Maturity Model*) [50],[22] se destaca.

A qualidade de software tem um papel importante em softwares com um arquitetura cliente-servidor pois neste tipo de arquitetura o processamento pode não ser feito em uma máquina somente, e ainda que seja feito na mesma máquina, é dividido, no mínimo, entre duas camadas, uma cliente e uma servidor, se estas camadas estiverem em máquinas diferentes vão estar fisicamente conectadas através de uma rede. Todas essas possibilidades aumentam a complexidade.

O *Gartner Group* [10], faz uma série de considerações sobre as diferenças das técnicas de teste tradicionais com relação ao teste de aplicações cliente-servidor. Por que é diferente testar aplicações cliente-servidor? A resposta simples e direta é porque o ambiente é mais complexo, consequentemente requer mais teste do que aplicações tradicionais. A dificuldade para se isolar um defeito é muito maior do que em uma arquitetura simples. Os

desenvolvedores precisam testar as qualidades técnicas e funcionais da aplicação, a usabilidade da interface, tempos de resposta e a precisão dos dados. Cada função do menu, cada objeto de cada janela tem que ser testado individualmente, combinado com as possíveis ações que diferentes usuários podem realizar em ordem totalmente alternada. Isso com o foco somente na aplicação cliente, desconsiderando as possíveis falhas que podem ocorrer na camada intermediária (*middleware).*

Segundo o *Gartner Group* [10], a maioria dos testes em aplicações cliente-servidor é executado de forma pobre, inconsistente e manual. Menos de 50% das organizações que desenvolvem aplicações usando ferramentas de teste cliente-servidor estão praticando alguma forma de metodologia. Para muitas organizações o teste inicia logo após a codificação e um pouco antes da instalação. Infelizmente este cenário freqüentemente leva a uma qualidade pobre e, consequentemente, eleva os custos de manutenção da aplicação, deixando os usuários de negócio descontentes.

Em um ambiente cliente-servidor é necessário testar todas as camadas, verificar o desempenho e confiabilidade quando executadas em rede ou sob uma carga maior de informação.

O *Quality Assurance Institute*, dos EUA, indica que "70% dos problemas de sistemas são de especificação, pois apenas 60% das informações formais e informais estão documentadas" [2], [65]. Essa margem de erro hoje é inaceitável em algumas áreas, como a medicina ou a engenharia de segurança, por exemplo. Outro fato a ser considerado é que se houver algum atraso no desenvolvimento de um sistema, normalmente a atividade de teste é uma das primeiras a ser enxugada, embora devesse ser a última a se comprometer, pois trata-se da validação de todo o esforço de gerenciamento e execução do projeto. Isto porque, normalmente, a fase de teste deveria consumir cerca de 45% de todo o tempo do projeto para obter-se qualidade no sistema, segundo o *Gartner Group* [10], e para se evitar essa situação, devem-se incluir as devidas metodologias de qualidade em sistemas, por exemplo, a atividade de testes deve ser paralela às demais etapas do projeto, como o verdadeiro ciclo de vida de testes, que acompanha o ciclo de vida da aplicação [2].

Infelizmente, poucas organizações de Tecnologia de Informação têm os especialistas, as ferramentas e os ambientes de teste apropriados para tratar de testes em grande escala. A maioria das modificações dos aplicativos é, no máximo, superficialmente testada. Em geral os dados de teste são criados a partir de um subconjunto dos dados de produção [18].

Apesar de poucas empresas adotarem uma metodologia de teste nos seus processos de desenvolvimento, existem muitos estudos sobre o assunto, nos quais são propostas

estratégias genéricas ou enfocando algum ambiente, com suas limitações e até ferramentas comerciais disponíveis.

Alguns desses trabalhos auxiliam apenas a elaboração do plano de teste, muitos não são completos no sentido de auxiliar a todas as etapas do desenvolvimento e do teste, outros são específicos para um ambiente e podem ser adaptados para cliente-servidor. Além desses trabalhos também encontram-se padrões com o objetivo de sistematizar a atividade de teste. Muitos destes trabalhos, estratégias e metodologias são também dependentes de ferramentas.

A estratégia de teste genérica mais conhecida é a de Pressman [52],[53] que sugere as etapas de unidade, integração, validação e sistema a serem adotadas durante o teste. Além disso, o teste envolve fases tais como planejamento, projeto e execução de casos de teste, e avaliação dos resultados obtidos. Mais recentemente essas etapas e fases têm sido também utilizadas em outros trabalhos que levam em consideração paradigmas de desenvolvimento espiral e interativo como é o caso do Processo Unificado da Rational (*Rational Unified Process* – RUP) [32] e da metodologia adotada pela ferramenta de teste da Rational, *Rational Suite Test* [57].

Com relação a trabalhos específicos para ambiente cliente-servidor destacam-se o trabalho do *Gartner Group* [10] que fez uma série de considerações a respeito do que uma estratégia de teste para este tipo de ambiente deveria se preocupar e também considera as fases do teste aplicadas ao paradigma espiral. O trabalho de Mosley [45] propõe diferentes testes para as camadas da arquitetura cliente-servidor. É crescente o interesse na utilização de um processo de teste padrão a fim de fornecer uma linha base para as atividades de teste tal como o Test-Rx [45].

1.2 MOTIVAÇÃO

Resumindo o que foi escrito acima, os seguintes itens podem ser ressaltados e servem como motivação para o presente trabalho:

1. importância do teste para o desenvolvimento de sistemas: o teste é uma atividade importante para a garantia de qualidade de software e custa caro;
2. muitos sistemas são entregues com defeitos por não serem testados adequadamente;
3. comprometimento das empresas com a garantia de qualidade no desenvolvimento e implantação de um programa desenvolvido por empresas de padronização ou que promovem a certificação como ISO e IEEE; nesse contexto teste é fundamental.

4. quantidade de componentes envolvidos com a tecnologia cliente-servidor, que dificultam o teste;
5. necessidade de uma metodologia para garantir a execução e controle de todos os passos necessários à execução da tarefa num ambiente cliente-servidor;
6. existência de metodologia, processo ou estratégia genéricos ou dependentes/associados a alguma ferramenta;

Esses itens mostram a necessidade de elaborar uma estratégia de teste de software adequada à tecnologia cliente-servidor, que seja prática, que possa ser aplicada em passos independentemente de ferramenta, e que norteie o desenvolvedor em todo o processo de desenvolvimento, contribuindo assim para aumentar a qualidade do teste e reduzir seu custo.

1.3 OBJETIVOS

Esse trabalho propõe uma estratégia de teste voltada para software cliente-servidor, denominada **ETACS** (**E**stratégia de **T**este de Software para **A**mbiente **C**liente-**S**ervidor).

A ETACS tem o objetivo de ser completa, no sentido de orientar os vários passos da atividade de teste ao longo do desenvolvimento do projeto com roteiros, até chegar na aplicação dos critérios e técnicas existentes, também pode ser aplicada independentemente de ferramentas.

A ETACS é resultado de estudos empíricos e teóricos. Primeiramente, um estudo empírico foi realizado em uma empresa cuja a maioria dos softwares desenvolvidos adota o modelo cliente-servidor. O estudo consistiu em avaliar, através de questionários e entrevistas, como a atividade de teste era realizada e quais as principais dificuldades encontradas. Numa segunda etapa, as estratégias de teste específicas ou não para o ambiente cliente servidor e diversos trabalhos da literatura foram estudados. Considerando os resultados empíricos, a estratégia ETACS foi proposta reunindo os pontos positivos de cada trabalho estudado e combinando-os numa série de passos.

A ETACS foi avaliada em um estudo de caso real na mesma empresa onde o estudo empírico foi realizado e resultados desses estudo de caso são também apresentados.

1.4 ORGANIZAÇÃO

No Capítulo 2, é dada uma descrição do ambiente para o qual essa estratégia está direcionada, como funciona a arquitetura cliente-servidor 3 camadas, as características que apresenta diferenças com outras arquiteturas conhecidas, particularidades de cada tipo, etc.

No Capítulo 3, é apresentada uma revisão bibliográfica das técnicas e critérios de cobertura de teste, e destaca-se a importância e ligação entre a qualidade e a área de teste de software.

No Capítulo 4 é apresentada uma revisão de alguns dos trabalhos publicados em literatura e no Capítulo 5, um resumo dos principais tópicos do relatório da pesquisa elaborada em uma empresa que utiliza dentre as suas tecnologias a arquitetura cliente-servidor. Estes itens formaram a base para a definição da estratégia, descrita no Capítulo 6.

Essa estratégia visa o ambiente de tecnologia cliente-servidor 3 camadas e foi utilizada em um estudo de caso em uma empresa que possui estas características, cujos resultados são apresentados no Capítulo 7.

No Capítulo 8 são apresentadas as conclusões e contribuições do trabalho bem como as preocupações com sua continuidade.

No APÊNDICE A é apresentada uma lista das ferramentas disponíveis comercialmente, que podem auxiliar a atividade de teste. No APÊNDICE B está relacionado o questionário, com objetivos e respostas, relacionado ao estudo empírico do Capítulo 5. Nos apêndices C, D, E, F e G são apresentados detalhes da realização dos cálculos necessários para a elaboração da estratégia, descrita no Capítulo 6 e na aplicação dela durante o estudo de caso, apresentada no Capítulo 7, bem como informações complementares e justificativas necessárias para o entendimento de alguns pontos. No ANEXO 1, são apresentados em detalhe os casos de usos utilizados para aplicação da estratégia descrita no Capítulo 6.

2 DESENVOLVIMENTO CLIENTE-SERVIDOR

2.1 INTRODUÇÃO

É importante conhecer como funciona este tipo de tecnologia, tudo que ela envolve, para que possa ser entendida a complexidade envolvida no ambiente e a necessidade de uma estratégia de teste para esse tipo de desenvolvimento, para auxiliar a vida do desenvolvedor e produzir software com qualidade minimizando as chances de fracasso.

Para compreender melhor os limites entre arquitetura de *mainframe*, arquitetura servidor de arquivo (*file-server)* e a variedade de arquiteturas cliente-servidor, segundo Herb Edelstein [13], pode-se dividir uma aplicação em seis componentes:

1. Serviços de apresentação: são fornecidos por um dispositivo que aceita as entradas do usuário e mostra tudo o que a lógica de apresentação mandar.
2. Lógica de apresentação: controla a interação entre o usuário e o computador através dos eventos gerados pelo usuário, por exemplo: programa o que deve acontecer ao selecionar uma opção de menu, clicar em um determinado botão, etc.
3. Lógica do negócio ou da aplicação: é um conjunto de decisões, cálculos e operações que a aplicação deve executar, tal como o cálculo do salário de um empregado, a análise da melhor forma de aplicar um recurso financeiro ou os procedimentos necessários para transferir fundos de uma conta bancária para outra.
4. Lógica de dados: é a especificação das operações que devem ser executadas no banco de dados para cumprir os requisitos da lógica do negócio. Para um banco de dados relacional, são declarações SQL, tais como *SELECT*, *UPDATE* e *INSERT*.
5. Serviços de dados: são as ações que o sistema gerenciador de banco de dados executa para que se cumpra a lógica dos dados. Isto inclui a definição de dados, a manipulação de dados, *commit* e *rollback* de transação, etc.

6. Serviços de arquivos: normalmente são funções do sistema operacional de leitura e gravação de dados.

Nos ambientes de *mainframe* o terminal fornece os serviços de apresentação, enquanto o *host* fornece o restante das funções. Isto causa basicamente dois problemas sérios. Primeiro, a interface com o usuário fica limitada a um terminal, ou seja, é difícil, se não impossível, fornecer uma interface GUI (*Graphical User Interface*) eficiente quando todos os processamentos residem na máquina central. Segundo, cada aplicação ou usuário adicional causa uma carga substancial ao *mainframe*. Isto causa uma perda de escalabilidade. A própria carga da rede é maior do que a necessária, uma vez que o *host*, além de enviar os dados para o terminal, tem que transmitir instruções para a formatação da tela [13].

A arquitetura *file-server* coloca os serviços de arquivos no servidor e todas as demais funções ficam alocadas na máquina cliente. O servidor de arquivos apenas recupera os arquivos necessários e os transmite para a estação cliente. Assim sendo, pode-se afirmar que a arquitetura *file-server* resolve os problemas da arquitetura baseada em *mainframe*, porém cria novas dificuldades. Resolve os problemas na medida em que permite a criação de interface gráfica eficiente e acrescenta pequena carga de trabalho à CPU toda vez que se aumenta o número de usuários ou de aplicações na rede. Em compensação, cria os seguintes inconvenientes: a demanda na estação cliente torna-se muito alta e aumenta consideravelmente o tráfego na rede [13].

A arquitetura cliente-servidor foi projetada para resolver esses problemas, separando os componentes de uma aplicação e colocando cada um deles no local onde é mais eficiente. Uma vez que existem diversas formas de distribuir esses componentes, é importante compreender as diferentes arquiteturas cliente-servidor e para que tipo de aplicações cada uma é adequada [13].

A Figura 2.1 retrata um exemplo da complexidade do ambiente cliente-servidor. Em um nível macro, há diferentes componentes: cliente (*client*), servidor (*server*) e meio (*middleware*) e pode ser facilmente percebido que o ambiente cliente-servidor é uma arquitetura multi-camada. Em um nível mais micro, há o aspecto da heterogeneidade dos clientes e servidores e a possibilidade de configurações diferentes nestas camadas, além da distribuição dos dados e da natureza da interface gráfica dirigida a eventos.

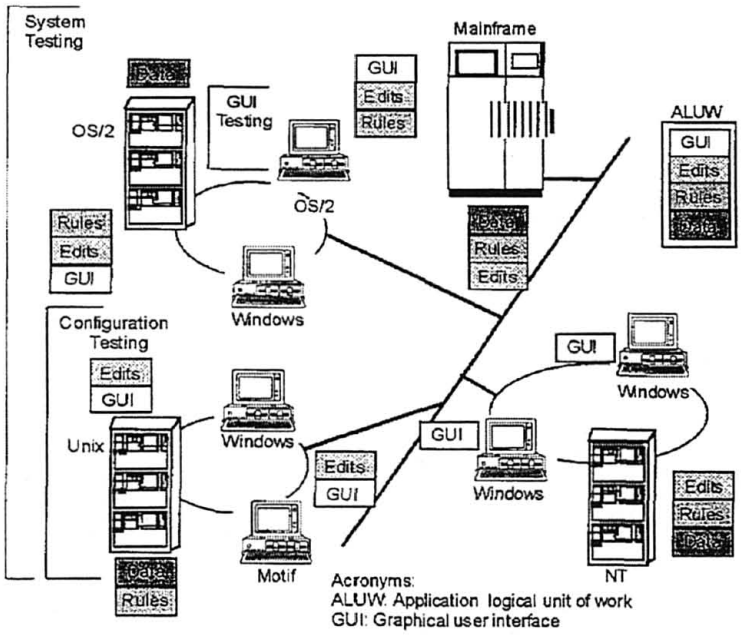


FIGURA 2.1 COMPLEXIDADE DO AMBIENTE CLIENTE-SERVIDOR

2.2 ARQUITETURAS

As arquiteturas cliente-servidor mais conhecidas e utilizadas são 2 e 3 camadas:

2.2.1 Arquitetura 2 camadas

A configuração cliente-servidor mais simples é um modelo duas-camadas (*two-tiered)*, na qual um cliente (camada de apresentação) solicita serviços a um servidor (camada de dados). É possível que um servidor funcione como cliente de outro servidor formando uma espécie de arquitetura cliente-servidor hierárquica. Neste caso, trata-se ainda de uma abordagem duas-camadas, porém, encadeada. [13]

No mínimo, uma arquitetura cliente-servidor pressupõe que os serviços de apresentação e a lógica de apresentação residam no cliente. Isto resolve os inconvenientes de uma interface tipo terminal fornecendo um gerenciador de interface gráfica local. Mas, para onde vão os demais componentes da aplicação? [13]

A primeira possibilidade é colocar os serviços de dados e de arquivos no servidor e, a lógica do negócio e de dados no cliente. Esta abordagem pode ser chamada de "SGBD remoto". A maioria das definições superficiais de computação cliente-servidor emprega este modelo: a aplicação está no cliente e o SGBD no servidor. Uma vez que este modelo coloca a menor demanda no servidor ele é o que provê a melhor escalabilidade.

No entanto, para aplicações complexas que envolvem uma grande interação com banco de dados, este modelo pode ocasionar sobrecarga na estação cliente e na rede. Outra desvantagem desse modelo diz respeito ao gerenciamento da aplicação, pois uma cópia completa do código executável reside em cada estação cliente. Quando o código é alterado, tem-se que substituí-lo em todas as estações [13].

Pode-se reduzir a carga do cliente e da rede transferindo parte da lógica do negócio para o servidor. Este modelo cliente-servidor é chamado de *split-logic*, que em português significa aproximadamente lógica distribuída.

O próximo passo para minimizar o impacto da aplicação sobre as estações cliente é transferir toda a lógica do negócio para o servidor, deixando apenas a lógica e os serviços de apresentação no cliente. Este modelo é chamado de apresentação remota [13], [48].

O mecanismo normalmente utilizado para implementar a lógica do negócio nos servidores são as *stored procedures*. Uma vez que elas são compiladas, o uso sensato de *stored procedures* pode melhorar o desempenho e reduzir a carga do servidor, se comparado ao uso de SQL dinâmico (comando SQL não compilado emitido pela estação cliente).

Para compreender porque deve usar *stored procedure* tem-se que entender os dois passos do processo de compilação de um comando SQL dinâmico. Primeiro o comando SQL é processado, o que significa checar nomes de tabelas e colunas, direitos de acesso do usuário que está acionando o procedimento e otimizar o comando SQL. Otimização é o processo através do qual o SGBD determina o melhor caminho de acesso aos dados. O segundo passo é compilar a parte procedural do código. Nota-se que, se não utilizar *stored procedures*, todas essas tarefas serão executadas em tempo real, durante o processamento da transação. Outro aspecto importante quanto ao desempenho é que uma *stored procedure* pode ser armazenada em um *cache* de procedimentos, reduzindo assim o número de entradas e saídas (*In/Outs - I/Os*) da aplicação [48].

As arquiteturas *two-tiered* (SGBD remoto, lógica distribuída e apresentação remota) também apresentam alguns problemas, considerando-se aplicações complexas, com muitos clientes e bases de dados heterogêneas [13], [48]. O primeiro problema é definir onde colocar a lógica do negócio. Colocar no cliente implica muito esforço no momento de uma alteração ou de um *upgrade*, pois o software terá que ser alterado, instalado e testado em todas as estações [13].

O segundo problema diz respeito ao uso de *stored procedures* para implementar lógica complexa. Infelizmente, as linguagens procedurais utilizadas nas *stored procedures* não têm capacidade equivalente às linguagens convencionais. Além disso, possuem um ambiente de

desenvolvimento precário no que diz respeito a testes, depuração, controle de versão e gerenciamento de bibliotecas. Pior que isso é o fato de todas as implementações de *stored procedures* utilizarem uma linguagem proprietária associada a um SGBD específico. Atualmente, as *stored procedures* não são portáveis entre os SGBDs. Isto significa que em se mudando o SGBD tem-se que rescrever toda a aplicação. Significa também que, se existem bases de dados heterogêneas, tem-se que escrever procedimentos customizados para cada SGBD [13], [48].

2.2.2 *Arquitetura 3 camadas*

A arquitetura três camadas (*three-tiered*) propõe-se a solucionar os problemas apresentados no modelo cliente-servidor duas camadas (*two-tiered*) [13]. Na teoria, os sistemas cliente-servidor 3 camadas são mais escaláveis, robustos e flexíveis. Além disso eles podem integrar dados de várias origens. Exemplos de sistemas 3 camadas: Monitores de Transação (*TP Monitors*), objetos distribuídos e a *Web* [48].

Nessa arquitetura, o cliente fica dedicado à lógica e aos serviços de apresentação, conhecido por isso como camada de apresentação. Esta aciona outra parte da aplicação que reside em uma camada intermediária, conhecida como camada de negócio ou camada de aplicação. A camada de negócio está em um servidor de aplicação que executa a lógica do negócio e solicita serviços à camada de dados que é responsável pela lógica dos dados. Na camada de dados encontra-se o gerenciador de banco de dados que fica dedicado aos serviços de dados e de arquivos, que podem ser otimizados sem o risco de usar *stored procedures*. O servidor de aplicação pode controlar transações e garantir integridade em bancos de dados distribuídos através do gerenciamento de *two-phase commit* heterogêneo [48].

O ambiente cliente-servidor, como visto na Figura 2.2 [48], é composto de:

1. uma caixa Cliente - com uma caixinha representando a Interface do Usuário Gráfica (*Graphical User Interface-GUI)/* Interface do Usuário Orientada a Objetos (*Object Oriented User Interface-OOUI)* responsável pela camada de apresentação dos dados para o cliente (*Visual Basic*, *Delphi*, *Sql Windows,* etc) e outra caixinha que representa outra forma de apresentar os dados para o cliente através da *internet:* o *Web Browser.*

2. uma caixa Servidor – que contém um componente DBMS, composto por sistemas gerenciadores de base de dados (*Sql Server*, *Sybase*, *Oracle*, etc); se houver mais de um servidor, com outros componentes tais como Monitores de transação (OLTP), servidor *Web*, etc.; passa a ser arquitetura 3 camadas.

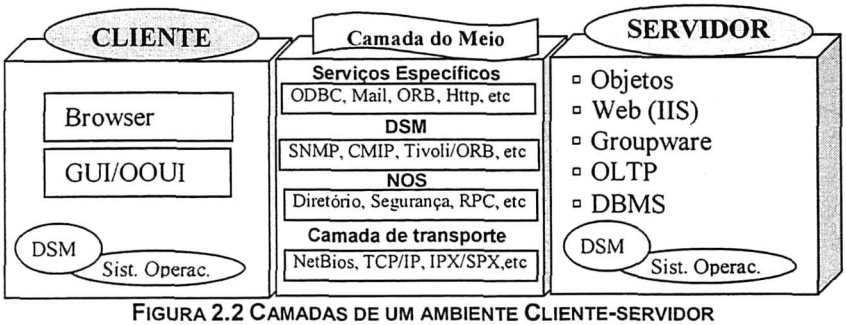


FIGURA 2.2 CAMADAS DE UM AMBIENTE CLIENTE-SERVIDOR

3. uma caixa *Middleware* - faz a conexão do cliente com o servidor e pode ter uma série de produtos e camadas ligando as duas camadas cliente e servidor.

Os limites entre lógica de apresentação, lógica do negócio e lógica dos dados são confusos. Por essa razão, a lógica do negócio pode aparecer em todas as três camadas. Deveria ser determinado o local de uma função específica usando critérios como facilidade de desenvolvimento e teste, facilidade de administração, escalabilidade dos servidores e desempenho, incluindo carga da rede e de processamento. Estes critérios freqüentemente são conflitantes entre si [13].

As aplicações cliente-servidor podem ser diferenciadas na forma como a aplicação distribuída será dividida entre o cliente e o servidor, como pode ser visto na Figura 2.3 [48]. No modelo com ênfase no servidor, a maior parte das funções fica no servidor, no modelo com ênfase no cliente o inverso acontece. Mas o importante é que o modelo três camadas não vai sobrecarregar nem um nem outro, pois o que estiver com maior sobrecarga pode ser manipulado para que funções executem no servidor de aplicação. Logo, verifica-se o gargalo e manipulam-se as funções para balancear a sobrecarga de alguma camada [48].

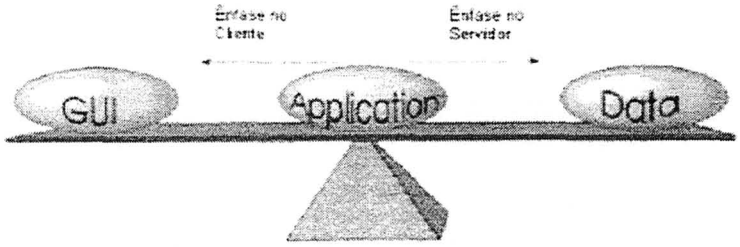


FIGURA 2.3 ARQUITETURA CLIENTE-SERVIDOR EM TRÊS CAMADAS

Segundo Bonifácio [7], este modelo apresenta as seguintes vantagens:

- Independência de código, ou seja, é muito mais fácil trocar uma camada inteira quando necessário, ou isola-se o código da alteração não comprometendo as demais camadas. Uma vez que se tenham todas as interfaces bem definidas entre as camadas, é fácil trocar a camada de lógica de banco de dados que está usando SQL Server 7 para uma com Oracle 8i, por exemplo. Esta característica é chamada de portabilidade.

- Independência de desenvolvimento e reutilização de código: uma equipe pode se concentrar em apenas desenvolver a parte complicada da aplicação, que é a lógica de negócio, sem se preocupar em criar formulários e códigos para validá-los. O desenvolvimento de boas interfaces requer habilidade e treinamento. Nem todo profissional que programa bem a lógica do negócio é capaz de escrever um bom código de apresentação [48].
- Facilidade de gerenciamento de projeto: uma vez que se tenha a aplicação bem projetada, com todas as interfaces definidas, é muito mais simples gerenciar grandes times de desenvolvimento trabalhando cada qual em um ponto da aplicação independentemente [7].
- Escalabilidade: facilidade de adicionar ou remover estações cliente com apenas um leve impacto no desempenho, isto é escalabilidade horizontal e migrar para uma máquina servidora mais rápida ou vários servidores, isto é, escalabilidade vertical [48]

Conforme Mosley [45] lembrou, o fato da arquitetura cliente-servidor estar rapidamente se tornando a abordagem de desenvolvimento preferida em muitas organizações, testadores de software devem reavaliar os *frameworks* de testes tradicionais.

2.3 Considerações Finais

Como foi observado, apesar de aumentar a complexidade, o modelo cliente-servidor proporciona várias vantagens como portabilidade e escalabilidade e, por isso, é altamente utilizado pelas empresas em geral no Brasil e no mundo [45], observado pelos congressos, artigos em revistas e comprovados pelo *Gartner Group* [10].

Claro que, aumentando a complexidade do ambiente, aumenta também a complexidade da atividade de teste, pois o número de variáveis que podem afetar e provocar um defeito é muito maior e também alguns testes que não seriam necessários num ambiente simples passam a ser essenciais neste tipo de arquitetura.

A utilização destas arquiteturas sem uma metodologia de teste para auxiliar esta atividade torna difícil, demorado e caro produzir um sistema de software com qualidade. Para estabelecer a estratégia deste trabalho foi considerado o ambiente cliente-servidor 3 camadas, descrito nesse capítulo por ser o mais abrangente.

3 RESUMO SOBRE TESTE DE SOFTWARE

3.1 INTRODUÇÃO

O IEEE tem realizado vários esforços de padronização, inclusive da terminologia utilizada no contexto de Engenharia de Software. O padrão IEEE número 610.12-1990 [28] diferencia os termos: defeito (*fault, bug*) é um passo, processo ou definição de dados incorreto (exemplo: uma instrução ou comando incorreto); engano (*mistake*) é uma ação humana que produz um resultado incorreto (exemplo: ação incorreta tomada pelo programador); erro (*error*) é uma diferença entre o valor obtido e o valor esperado, ou seja, qualquer estado intermediário incorreto ou resultado inesperado na execução do programa constitui um erro; e, falha (*failure*) é uma produção de uma saída incorreta com relação à especificação[38]. Neste trabalho, os termos engano, defeito e erro serão referenciados como defeito (causa) e o termo falha (conseqüência) como um comportamento incorreto do programa.

Segundo Myers [46]: *"Teste é o processo de executar um programa ou sistema com a finalidade de encontrar erros"*. Com a descrição do ambiente cliente-servidor apresentada no Capítulo 2 e conforme o exemplo da Figura 2.1 é possível vislumbrar a complexidade de encontrar defeitos em um programa que use esta tecnologia, por causa da quantidade de variáveis envolvidas no ambiente, tipo de servidor, protocolo de rede, etc.

A melhoria da qualidade de serviços vem sendo perseguida pelas diversas entidades que atendem a um público cada vez mais variado, seja pelo interesse em aumentar os lucros (como empresas privadas) ou pela necessidade de se adequar a um orçamento cada vez mais enxuto (como empresas públicas) [67]. Compreender o que está acontecendo em uma organização específica e em seus projetos de software é crucial para que se possa planejar, controlar e melhorar o desenvolvimento de software [72].

Diante do contexto globalizado do mercado de software, a garantia da qualidade no processo de desenvolvimento tem se tornado, cada vez mais, uma premissa básica para a competitividade das indústrias de software. Existe uma série de certificações e

padronizações para a área de desenvolvimento de software. Vários modelos têm sido propostos por instituições no mundo inteiro, destacado-se o modelo SEI/CMM (Software *Engineering Institute/Capability Maturity Model*) da Universidade Carnegie Mellon [50], [22], [35]. O modelo CMM permite que a organização saiba em que nível de maturidade está seu processo de software e que realize determinadas práticas de melhoria com o objetivo de atingir níveis superiores e a busca de uma melhoria contínua [66]. O modelo CMM tem 5 níveis de maturidade sobre o controle do processo. Cada nível é associado a um conjunto de *Key Practice Areas* – KPAs.

As metodologias propostas SEI/CMM, ISO/SPICE e *Institute of Electrical and Electronics Engineers* - IEEE (como exemplo STD1008 Padrões para teste de unidade [26] e STD829 Padrões para documentação de Teste de Software [25]) não enfatizam apenas o processo gerencial do projeto de desenvolvimento ou subcontratação, mas também fases e ambientes específicos para validação do projeto ou testes dos resultados obtidos durante o projeto [2]. A estratégia proposta leva em consideração estas organizações oficiais de padronização e procura estar adequada com as normas a que dizem respeito [23],[24],[25],[26],[27],[28].

Por que testar não é simples? Porque para testar com eficiência é preciso conhecer os sistemas a fundo e os sistemas não são, em geral, simples nem fáceis de entender. Segundo Vyssotsky [71], "*primeiro e mais importante, se eu não sei o que o programa deve fazer, nenhum teste pode me garantir que ele está fazendo aquilo*". É difícil porque exige habilidade, conhecimento e experiência para que sejam obtidos bons resultados. Hetzel [20] aponta 4 fatores como essenciais para a atividade de teste: 1) a criatividade e a inteligência; 2) o conhecimento funcional; 3) a experiência na realização de testes; e 4) a metodologia de teste.

Outro detalhe importante é o custo do teste; Knowles [31] salienta que o custo do teste é significativo tecnologicamente mas não adiciona nada visível ao produto. Este custo é justificado pela confiança na existência de um nível de aceitação de qualidade e desempenho ou uma indicação de que alguns tipos de defeitos estão ausentes. O rigor e o custo envolvidos nessa atividade dependem do impacto da ocorrência de falhas no sistema [58].

Segundo Pressman [52], a atividade de teste corresponde a 30% e 40% do custo total do software. Devido a esta importância, o teste deve ser feito da maneira correta para que um resultado positivo seja alcançado. Um bom teste é aquele capaz de revelar a presença de defeitos no programa [46].

É importante observar também que o efeito total de corrigir defeitos muito tarde aumenta os custos de reparo. Quanto maior o atraso entre o defeito ser introduzido e detectado, maior será o custo da correção [15].

Estudos e registros históricos apontados por Simões [61], relatam que considerando-se a produtividade de um projeto de sistemas em termos de número de pontos por função por unidade de tempo, os percentuais de tempo despendidos nas fases de um projeto podem ser, em média, distribuídos conforme a Figura 3.1.

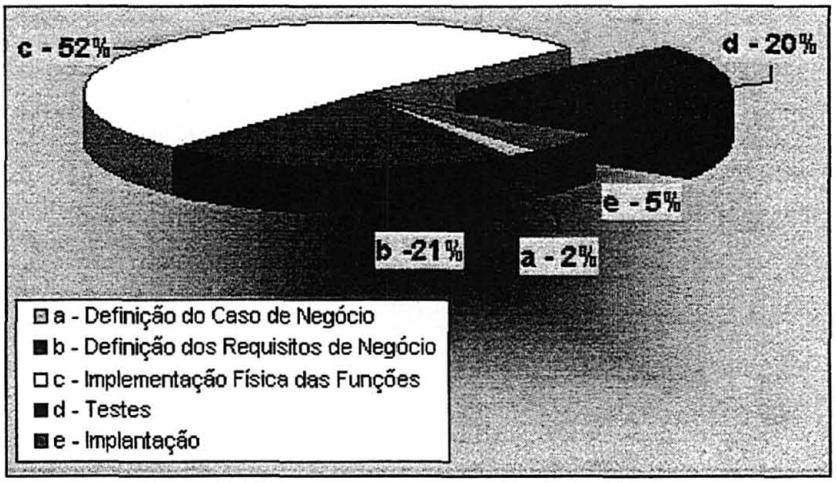


FIGURA 3.1 TEMPO DESPENDIDO NAS FASES DO PROJETO

Observa-se na Figura 3.1 que o tempo de teste em relação ao tempo total do projeto corresponde, em média, a 20%. Aplicando-se esse percentual ao custo total de desenvolvimento de um projeto, desconsiderando-se o custo de correção dos problemas encontrados, nota-se que essa é uma parcela significativa do custo total do projeto.

Um projeto normalmente está subordinado a um cronograma. O gerente do projeto juntamente com os líderes responsáveis estão constantemente procurando uma oportunidade de recuperar o tempo perdido. Se a pressão do tempo aumenta, os argumentos para omissão de atividades de teste ficam cada vez mais fortes. Eles acabam se convencendo de que estão calculando os riscos apropriadamente, mas esta abordagem leva à resultados pobres [3].

Além da questão do custo e do tempo que se gasta com testes tem-se uma série de vantagens em se aplicar testes adequadamente e com qualidade:

- reputação de mercado da empresa é afetada, em detrimento das experiências ruins dos clientes com software de má qualidade, com muitos defeitos;
- tempo gasto na correção dos defeitos na manutenção é um tempo não gasto no desenvolvimento de novos produtos, levando a perder posição de mercado;

- clientes tenderão a desconsiderar sua empresa num processo de licitação devido à falta de qualidade do produto entregue anteriormente ou, ao contrário, podem considerá-lo com maior peso por produtos de boa qualidade já comprados;
- é necessário para certificação de gerenciamento de qualidade reconhecidos, tal como ISO 9001 e fazendo o processo de teste repetitível, satisfaz-se um dos requisitos para o nível 2 do CMM e definindo o processo de teste satisfaz-se o nível 3 do CMM [45];
- a Legislação faz a empresa se responsabilizar por qualquer efeito da falha de software, independente de a falha ser devido a uma negligência ou não;
- a credibilidade de uma empresa ou de um produto é construída lentamente e por um longo tempo. Mas uma falha séria no software pode destruir isso rapidamente;
- os efeitos de falhas de software dependem do sistema em que eles ocorrem, mas podem incluir inconveniência, aborrecimento, desinformação, perda de informação, perda de dinheiro, mágoa pessoal e até morte.
- utilizando uma metodologia ou um processo de teste que tenha credibilidade garante-se que os componentes de alto risco do sistema são testados;
- evita problemas de omissão, testar inadequadamente partes do sistema ou de concessão, testar partes do sistema que não são importantes ou que resultam em testes redundantes [45];

Aplicar planejamento, métodos e métricas é tão importante quanto saber ouvir o que o usuário está querendo dizer [51]. "*Devemos planejar o teste como um general organiza sua batalha. Afinal mesmo você não vendo, os defeitos estão lá na aplicação, aguardando o momento certo de aparecer*". Esta frase foi colocada num grupo de discussão (*SQA user mailing list*) e reflete todo o problema de testar [51].

As principais técnicas para geração de dados de teste visam gerar dados que possam detectar a maioria dos defeitos com o mínimo de esforço e tempo. Algumas dessas técnicas estão descritas na próxima seção.

3.2 TÉCNICAS E CRITÉRIOS DE TESTE

Geralmente associados a uma técnica de teste estão diferentes critérios de teste. Um critério de teste é um predicado a ser satisfeito para se considerar a atividade de teste terminada. Ele, além de auxiliar a etapa de geração, fornecendo diretrizes para geração dos

Os critérios estruturais mais conhecidos são:

↳ **Todos-Nós** [54] ou **Cobertura de declaração de código** [15] - exige que a execução do programa passe, ao menos uma vez, em cada vértice do grafo de fluxo, ou seja, que cada comando do programa seja executado pelo menos uma vez. Cobertura de declaração é o número de declarações exercitadas pelo conjunto de teste, divididos pelo número total de declarações no módulo sendo medido. Em um programa de 100 declarações (executáveis), um conjunto de testes que atualmente usaram 75 destas declarações alcançariam 75% da cobertura de declarações.

↳ **Todos-Arcos ou Todos-Ramos** [54] ou **Cobertura de decisão** [15],[54] - requer que cada aresta do grafo, ou seja, cada desvio de fluxo de controle do programa, seja exercitado pelo menos uma vez; Cobertura de decisão ou ramo é similar a cobertura de declaração, mas em vez do número de declarações serem contadas, são contados o número de ramos equivalentes ou decisões. Uma declaração IF tem dois ramos, o ramo verdadeiro e o ramo falso. Declarações CASE e laços são equivalentes para ramos, desde que eles possam ser expressos como declarações IF. Se um programa contém 20 ramos, e um conjunto de teste exercita 10 deles, então 50% de cobertura de ramos tem sido alcançado.

↳ **Todos-Caminhos** - requer que todos os caminhos possíveis do programa sejam executados [54], [15]. Em qualquer programa que executa um laço um número variável de iterações, há um número infinito de caminhos diferentes através do programa, logo cobertura de todos os caminhos é geralmente impraticável.

↳ **Cobertura de condição-decisão** [15] – Cobertura de condição-decisão é realizada exercitando ambos os resultados verdadeiros e falsos de cada condição com uma decisão. Uma decisão composta com três comparações alcançaria cobertura de ramos ou decisão exercitando apenas os resultados verdadeiros e falsos da decisão, assim se necessita apenas dois casos de teste. Para se alcançar cobertura de condição-decisão, cada um dos três componentes usados para determinar a decisão também precisa de um caso verdadeiro e falso; isto pode ser feito em dois casos (por exemplo, todos verdadeiros, todos falsos). Cobertura de condição-decisão pode ser alcançada sem alcançar cobertura de decisão.

↳ **Cobertura de combinação de condição decisão** [15] – Não é apenas cada condição avaliada para verdadeiro ou falso, mas todas as combinações possíveis de condição são avaliadas para alcançar cobertura da combinação de condição-decisão. Por exemplo, se há três condições na decisão, oito casos de teste seriam necessários para a combinação das três condições. Se houver oito condições, então 256 casos de testes seriam necessários.

100% da cobertura de combinação de condição garantem 100% da cobertura de declaração ou ramos (para linguagens estruturadas modernas).

↳ Podem ser utilizados outros critérios estruturais [38], tais como: Cobertura de Condição [52]; Cobertura de Condições Múltiplas; o critério Boundary-Interior [21]; Cobertura LCSAJ (*Linear Code Sequence And Jump*) [73], [15].

3.2.1.2 Critérios baseados em fluxo de dados

Utilizam informações do fluxo de dados do programa para determinar os requisitos de teste. Esses critérios exploram as interações que envolvem definições de variáveis e referências a tais definições para estabelecerem os requisitos de teste [55].

Exemplos dessa classe de critérios são os critérios definidos por Rapps e Weyuker [55], [56], requerem a execução de caminhos a partir da definição da variável até onde ela foi utilizada. Uma definição da variável ocorre quando um valor é armazenado em uma posição de memória. Um uso de uma variável ocorre quando é feita uma referência a essa variável. O uso da variável pode ser computacional (c-uso), quando é usada em uma computação, soma por exemplo (lado direito da atribuição) e predicativo (p-uso), quando usada em uma condição. Os principais critérios são:

↳ **Todas-Definições** - requer que cada definição de variável seja exercitada pelo menos uma vez, não importa se por um c-uso ou por um p-uso.

↳ **Todos-Usos** - requer que todas as associações entre uma definição de variável e seus subseqüentes usos (c-usos e p-usos) sejam exercitadas pelos casos de teste, através de pelo menos um caminho livre de definição, ou seja, um caminho onde a variável não é redefinida.

↳ **Todos-Du-Caminhos** – exige que todos os du-caminhos do programa sejam cobertos. Um du-caminho c.r.a x, é dado por uma seqüência de nós $(n_1,...,n_j, n_k)$ onde n_1 possui uma definição de x e n_k possui um uso de x e $(n_1,...,n_j, n_k)$ é um caminho livre de definição c.r.a x (ou ainda, (n_j, n_k) possui um uso de x em um predicado e $(n_1,...,n_j, n_k)$ é um caminho livre de definição c.r.a x e $(n_1,...,n_j)$ é um caminho livre de laços).

Baseados nos critérios de Rapps e Weyuker [55], Maldonado [36] definiu os critérios Potenciais usos, introduzindo o conceito de potencial-associação. Esses critérios não exigem o uso explícito de uma variável para que uma associação seja estabelecida. Esses critérios são:

↳ **Todos-potenciais-usos** [36], [37] – esse critério é satisfeito se, para todo nó *i* e para toda variável *x* para a qual existe uma definição em *i*, pelo menos um caminho livre de definição

c.r.a x do nó i para todo nó e para todo arco possível de ser alcançado a partir de i seja executado.

↳ Outros critérios baseados em fluxo de dados que podem ser utilizados: a família de critérios K-tuplas requeridas de Ntafos [47]; Todos-potenciais-du-caminhos [36], Todos-c-usos e alguns p-usos; Todos p-usos e alguns c-usos; Todos c-usos; Todos p-usos; [55] Todos potenciais-usos/du [36], entre outras.

3.2.1.3 Critérios Baseados na Complexidade [52] [58]

Utilizam informações sobre a complexidade do programa para derivar os requisitos de teste. Um critério bastante conhecido dessa classe é:

↳ **Critério de McCabe ou Todos Caminhos Linearmente independentes** [42]– que utiliza a complexidade ciclomática do grafo de fluxo do programa, obtidas a partir das regiões mostradas na Figura 3.2, para derivar os requisitos de teste. Essencialmente, esse critério requer que um conjunto de caminhos linearmente independentes do grafo de programa seja executado, esse teste também conhecido como teste do caminho básico.

Um problema relacionado ao teste estrutural é a impossibilidade, em geral, de se determinar automaticamente se um caminho é ou não executável, ou seja, não existe um algoritmo que dado um caminho completo qualquer decida se o caminho é executável e forneça o conjunto de valores que causam a execução desse caminho [68]. Assim, é preciso a intervenção do testador para determinar quais são os caminhos não executáveis para o programa sendo testado. É desperdiçado tempo e esforço determinando-se elementos não executáveis [68], problema encontrado na aplicação de critérios.

3.2.2 Técnica funcional

É também conhecida como técnica de caixa preta, pois não leva em consideração como o programa foi implementado. Os dados de teste são derivados a partir da especificação e dos requisitos de software.

Exemplo de critérios funcionais:

↳ **Particionamento em classes de equivalência** [52] - a partir das condições de entrada de dados identificadas na especificação, divide-se o domínio de entrada de um programa em classes de equivalência válidas e inválidas. Em seguida seleciona-se o menor número possível de casos de teste, baseando-se na hipótese de que um elemento de uma dada classe seria representativo da classe toda, sendo que para cada uma das classes inválidas

deve ser gerado um caso de teste distinto. Por exemplo, se os números 0 a 99 são válidos, então todos os 99 números deveriam ser manuseados da mesma forma pelo software. Se o programa trabalha corretamente para o valor 7, por exemplo, ele provavelmente trabalhará também corretamente para 2, 35, 50, etc. Todos os valores de 0 a 99 estão na mesma classe de equivalência.

- **Análise do valor limite** [52] - é um complemento ao critério particionamento em classes de equivalência, sendo que os limites associados às condições de entrada são exercitados de forma mais rigorosa; ao invés de selecionar-se qualquer elemento de uma classe, os casos de teste são escolhidos nas fronteiras das classes, pois para esses pontos estão associados grande número de defeitos. O espaço de saída do programa também é particionado e são exigidos casos de teste que produzam resultados nos limites dessas classes de saída.

- **Grafo de causa e efeito** [52] [5] - os critérios anteriores não exploram combinações das condições de entrada. Este critério estabelece requisitos de teste baseados nas possíveis combinações das condições de entrada. Primeiramente, são levantadas as possíveis condições de entrada (causas) e as possíveis ações (efeitos) do programa. A seguir é construído um grafo relacionando as causas e efeitos levantados. Esse grafo é convertido em uma tabela de decisão a partir da qual são derivados os casos de teste.

- Podem ser utilizados outros critérios como testes de transição de estados entre outros.

3.2.3 Técnica baseada em erro

Utiliza certos tipos de defeitos comuns em programação para derivar requisitos de teste. Os casos de teste gerados são específicos para mostrar a presença ou ausência desses defeitos. A ênfase da técnica está nos enganos que o programador ou projetista pode cometer durante o desenvolvimento e nas abordagens que podem ser usadas para detectar a sua ocorrência.

Exemplos de critérios baseados em erros:

- **Estimativa de erro** [15]- Um bom testador freqüentemente tem um senso de onde os erros complicados devem estar escondidos e pode deduzir os melhores casos de teste para encontrar aqueles erros. Esta intuição é baseada em experiências anteriores ou em suposições comuns feitas pelos desenvolvedores. Exemplos de bons casos de teste são: divisão por zero, um arquivo, registro ou campo vazio, números negativos, caracteres alfabéticos para campos numéricos, ponto decimal, vírgula embutida, tamanhos máximos e

mínimos. Os casos de testes considerados desnecessários porque o desenvolvedor ou o projetista julgam que nunca vão acontecer, provavelmente sejam bons casos de teste.

↳ **Semeadura de Erros** (*Error Seeding*) [8] [58] – este critério exige que alguns erros sejam incluídos no programa, para que seja verificado durante o teste quais erros foram inseridos (semeados) no programa e quais são naturais. A idéia é verificar a relação entre o número de erros semeados revelados e o número de erros naturais revelados e obter uma indicação do número de erros naturais restantes.

↳ **Análise de mutantes** [12], [70] [58]: O critério Análise de Mutantes utiliza um conjunto de programas ligeiramente modificados (mutantes) obtidos a partir de um determinado programa P para avaliar o quanto um conjunto de casos de teste T é adequado para o teste de P. O objetivo é determinar um conjunto de casos de teste que consiga revelar, através da execução de P, as diferenças de comportamento existentes entre P e seus mutantes.

3.3 RELAÇÃO DE INCLUSÃO

A relação de inclusão é uma importante propriedade dos critérios, sendo utilizada para avaliá-los, do ponto de vista teórico. O critério Todos-Ramos, por exemplo, inclui o critério Todos-Nós, ou seja, qualquer conjunto de casos de teste que satisfaz o critério Todos-Ramos também satisfaz o critério Todos-Nós, necessariamente. Quando não é possível estabelecer essa ordem de inclusão para dois critérios, como é o caso de Todas-Defs e Todos-Ramos, diz-se que tais critérios são incomparáveis [38]. A Figura 3.3 mostra a relação de inclusão entre alguns critérios [56], [14], [37], baseados em fluxo de dados e controle.

Para entender como funciona a relação de inclusão, será apresentado um exemplo utilizando o critério Todos-Nós e Todos-Ramos. O Critério Todos-Ramos (ou de Arcos) inclui Todos-Nós, como apresentado na Figura 3.3. Por exemplo, considere o código simplificado da Figura 3.2 (Seção 3.2.1). Transformando este pequeno código em grafo de fluxo, cada instrução representa um nó como apontado ao lado do código e mostrado nesta figura.

Para alcançar cobertura de nós apenas um teste é suficiente, se o dado de teste for A=1 e B=500, cobre-se os nós 1, 2, 3, 4 e 5, ou seja, todos os nós do grafo. Mas é claro que isso foi apenas uma possibilidade. Nesse caso só é testado o código emitindo as mensagens 1, 2 e 3. Para satisfazer o critério Todos-Ramos falta cobrir os ramos (2,4) e (1,5). Se for criado mais um dado de teste, A=11 e B=251, o código será executado passando-se pelos nós 1 e 5 e pelo ramo (1,5). Ainda assim falta um ramo. É preciso criar um dado de teste A=1 e B=1 para passar também por este ramo. Ou seja, foram necessários 3 dados de teste para cobrir

o critério Todos-Ramos e apenas um destes 3 dados foi necessário para cobrir o critério Todos-Nós.

Outro aspecto interessante de conhecer esta relação de inclusão é que para os testes serem mais rigorosos devem ser utilizados critérios mais exigentes. E outro aspecto é que ao aplicar um critério mais abrangente, pode-se considerar que os critérios menos rigorosos foram também satisfeitos.

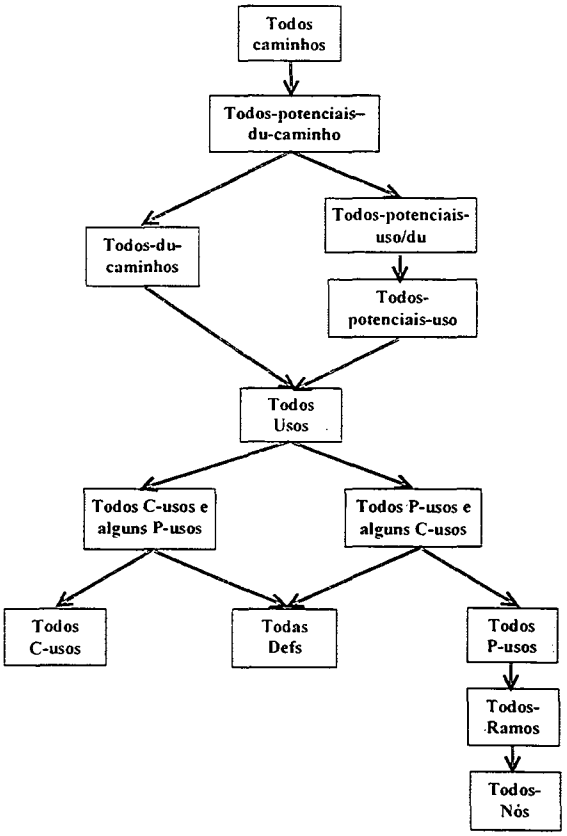


FIGURA 3.3 RELAÇÃO DE INCLUSÃO COM CRITÉRIOS BASEADOS EM FLUXO DE DADOS E CONTROLE

A idéia de cobertura pode ser estendida e ser aplicada em qualquer nível de abstração do sistema. Por exemplo, o número de opções de menu exercitados dão uma visão de cobertura baseada na tela. O número de funções de negócio exercitadas pode dar uma visão orientada ao negócio da qualidade do teste. A porcentagem de chamadas de submódulos em uma árvore de chamadas é uma medida de cobertura para sistema e subsistema (ferramentas comerciais podem fornecer isso). Mosley [45] apresenta uma discussão mais completa das métricas para análise de cobertura. Quanto teste é suficiente? Qual o critério de parada de teste? Hetzel fornece o guia clássico do quanto testar; e quanto testar depende diretamente do risco [20].

Ferramentas de software ajudam muito na avaliação da cobertura e na execução de vários tipos de testes. Existem várias disponíveis comercialmente, conforme descrito na próxima seção.

3.4 FERRAMENTAS

Com o aparecimento de ferramentas de suporte ao teste, no final da década de 80 e início de 90, houve um desenvolvimento mais significativo na área de teste. Durante alguns anos, a atividade formal denominada "teste de software" perdeu um pouco de espaço como instrumento de garantia de qualidade, ocorrendo a valorização das atividades preventivas, como é o caso das inspeções e revisões por pessoas qualificadas (*peer-reviews*). Atualmente a evolução das ferramentas para suporte ao teste vêm mudando este quadro, como pode ser observado na Figura 3.4, que mostra a avaliação feita em 1996, os dados de 96 a 99 são estimativas, fazendo com que o teste de software volte a despertar o interesse das grandes organizações. Mas, é claro, teste de software não é uma coisa nova [1].

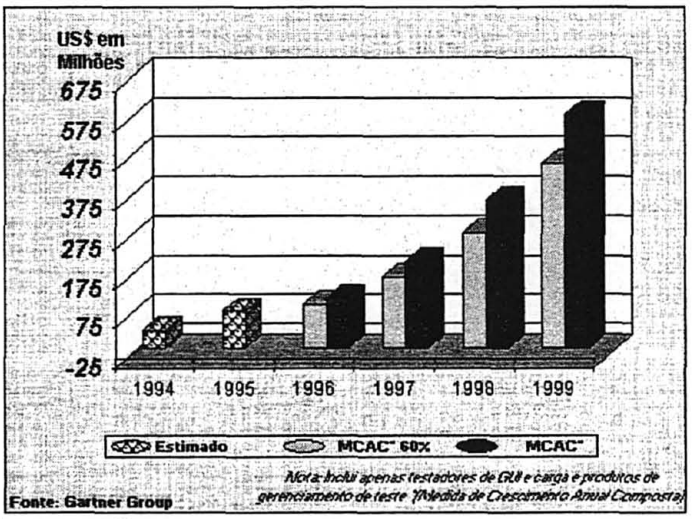
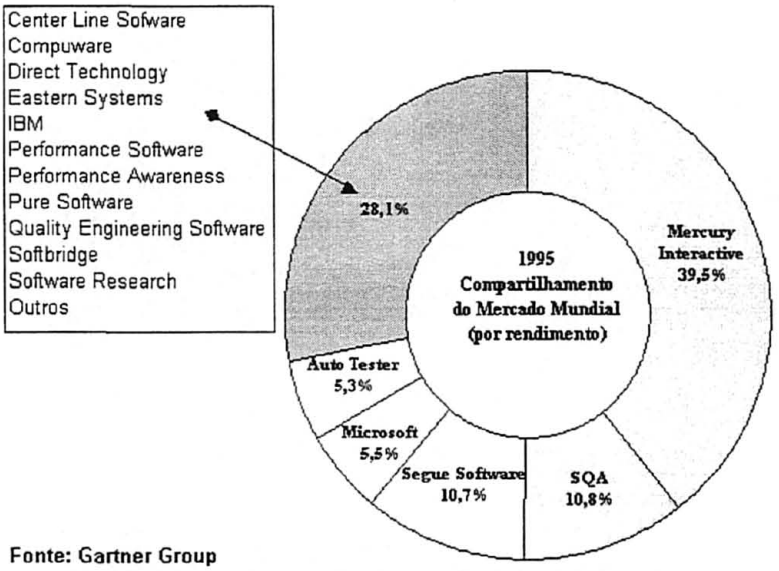


FIGURA 3.4 MERCADO DE FERRAMENTAS DE TESTE AUTOMATIZADAS CLIENTE-SERVIDOR NO MUNDO

Dados fornecidos pelo *Gartner Group* [10] revelam que o mercado de ferramentas de teste gerais é estimado em aproximadamente 500 milhões de dólares, composto do rendimento de várias indústrias que produzem ferramentas para depuração, analisadores de falhas, detectores de erros de execução, analisadores de cobertura e geradores de dados de teste em todas as plataformas.

A Figura 3.5 apresenta a distribuição do rendimento pelas empresas pesquisadas. Embora estas informações estejam desatualizadas, elas mostram efetivamente o crescimento das ferramentas nos últimos anos. Algumas empresas uniram-se para ganhar mais força como é o caso da *Pure* Software e *Atria* Software, a *Compuware* adquiriu a *Dirtect Technology*, a *Mercury* adquiriu a *Blue Lagoon*, a *Center Line* adquiriu a *Veritas* Software e muitas outras

compras e uniões [10], provando que este mercado tem despertado interesse nas grandes e médias empresas de desenvolvimento de software. E como mostra a Figura 3.4 os valores monetários envolvidos (500 milhões de dólares) são bastante expressivos.



Fonte: Gartner Group

FIGURA 3.5 COMPARTILHAMENTO DO MERCADO MUNDIAL EM 1995 POR RENDIMENTO

Se o teste é bem controlado e efetivo mas usa uma grande quantidade de recursos humanos, o uso de ferramentas de teste para automatizar partes do processo de teste, pode reduzir o custo dessa atividade [10]. Além disso:

- testes com melhor qualidade aumentam a produtividade;
- testes podem ser executados sem acompanhamento, à noite;
- automatizando tarefas repetitivas, é possível realocar melhor o pessoal;
- quando o software é alterado, os testes feitos antes da alteração podem ser refeitos automaticamente (teste de regressão);
- ferramentas auxiliam a detecção de erros difíceis;
- pode-se reduzir em até 80% os custos do teste usando-se uma ferramenta;
- mais testes poderão ser feitos em menos tempo;
- qualidade pode ser medida através de coberturas.

Além de reduzir o custo, a automação dos testes elimina os efeitos das diferenças individuais [19]. Testadores são únicos e eles fazem com que seu trabalho seja carregado das suas características pessoais. Se uma pessoa é mais detalhista vai fazer determinados testes que uma pessoa mais prática não faria e vice-versa. Isto pode resultar em vastos níveis diferentes de efetividade, causados pela educação, treinamento e experiências dos

testadores assim como pela influência do seus hábitos de trabalho. Automatizar o processo de teste força os testadores a seguir um procedimento repetitível. Este processo então torna-se um, que é consistente para todos os testadores, porque eles são forçados por uma ferramenta a executarem estes passos.

Os testes devem ser automatizados, pelo menos nas fases em que isto é possível. Existem muitas ferramentas que possibilitam essa automatização para o ambiente cliente-servidor. Para se ter uma idéia, observa-se no APÊNDICE A, vários exemplos de ferramentas de teste disponíveis comercialmente que dão suporte a uma série de testes importantes e necessários. Algumas das empresas apresentadas fornecem mais produtos além dos apresentados e também existem no mercado outras empresas que disponibilizam ferramentas similares.

3.5 CONSIDERAÇÕES FINAIS

A qualidade do software é um item que tem sido buscado pela maioria das empresas, e tem sido um diferencial de mercado as que têm alcançado reconhecimento de organizações oficiais de padronização, porque isto aumenta a confiabilidade da empresa junto aos clientes. A atividade de teste é altamente importante para melhorar a qualidade do software. Os produtos melhor testados provavelmente terão menos problemas e maior chance de sucesso e satisfação do cliente.

Existem na literatura vários tipos de critérios de teste, sendo impossível afirmar que um deles é melhor que o outro. Na realidade eles são completares. Existe uma relação de inclusão entre alguns dos critérios. Estes critérios são baseados na funcionalidade, no código e nos erros e encontram diferentes classes de defeitos nos sistemas.

Este capítulo é de grande relevância nesse trabalho, pois relaciona vários critérios de testes já conhecidos na literatura que podem ser aplicados nos sistemas ou parte deles conforme a necessidade identificada pelas características do sistema. A estratégia proposta (ETACS) não define um critério novo, mas faz algumas considerações para a aplicação dos já existentes, tendo em vista o ambiente cliente-servidor.

As ferramentas de teste podem auxiliar muito a atividade de teste e eliminar muito do trabalho manual e repetitivo, diminuindo a chance de errar na aplicação dos testes. A ETACS não está vinculada a uma ferramenta mas ressalta que é muito interessante a utilização de ferramentas para auxiliar a atividade em testes específicos, como é o caso de teste de carga e desempenho, por exemplo.

4 TRABALHOS RELACIONADOS

Uma estratégia de teste de software integra técnicas de projeto de casos de teste numa série bem definida de passos que resultam na construção bem sucedida do software. Esses passos podem envolver uma combinação das técnicas e aplicação de diferentes critérios de teste.

Para se desenvolver uma atividade de teste de maneira a atingir o objetivo de qualidade que se espera, não existe um teste único a ser realizado e sim uma combinação de vários testes ao longo do projeto.

Uma estratégia deve definir cada etapa a ser cumprida. Uma estratégia define então quais as etapas que serão executadas e como será realizada cada etapa, quais métodos, critérios e técnicas deverão ser utilizados para cumprir os passos da atividade de teste. Conforme descrito por Pressman [52], "*um grande número de estratégias de teste de* software *tem sido proposto na literatura*".

Neste capítulo serão apresentadas as estratégias de teste específicas ou não para o ambiente cliente servidor e diversos trabalhos da literatura que foram estudados. A estratégia ETACS foi proposta reunindo os pontos positivos de cada trabalho estudado e combinando-os numa série de passos. O objetivo não é esgotar o assunto e nem compará-los mas apenas abordar os trabalhos base para a estratégia.

4.1 ESTRATÉGIA DESCRITA POR PRESSMAN

Uma estratégia de teste bem conhecida descrita por Pressman [52], mostrada na Figura 4.1, relaciona algumas etapas de testes com as próprias fases de desenvolvimento do software.

Inicialmente os testes focalizam cada módulo individualmente, garantindo que ele funcione adequadamente como uma unidade. O teste de unidade faz uso em geral, da técnica estrutural de caixa branca, exercitando caminhos específicos da estrutura de controle de um

módulo. Nesta etapa devem ser testadas: estrutura de dados locais, condições de limite, etc. [52].

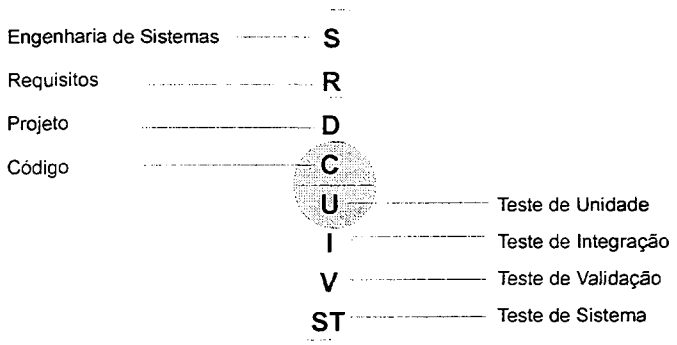


FIGURA 4.1 ESTRATÉGIA DE TESTE

Em seguida, os módulos devem ser montados ou integrados para formarem um pacote de software. O teste de integração visa a construir o programa a partir dos módulos testados no nível de unidade, realizando-se ao mesmo tempo, testes para descobrir defeitos associados a interfaces entre os módulos. Para isto a técnica funcional é mais utilizada, contudo alguns critérios estruturais são utilizados para garantir a cobertura de caminhos importantes de controle. Há uma tendência de se testar todos os módulos combinados como um todo, mas a correção é difícil porque o isolamento das causas é complicado. O mais correto é usar uma abordagem incremental que pode ser *top-down* e *bottom-up*. Maiores detalhes sobre esse tema podem ser achados em [52].

Depois que o software foi integrado (construído), um conjunto de testes de alto nível é realizado. O teste de validação verifica se o software atende a todas as exigências funcionais, comportamentais e de desempenho estabelecidas durante a fase de análise. Para cumprir esta etapa são mais adequados os critérios funcionais para obtenção dos dados de teste. Nesta etapa são realizados testes alfa e beta e uma revisão de configuração para verificar se tudo que é necessário foi desenvolvido.

A última etapa de testes de alto nível combina o programa a ser testado com outros elementos: *hardware*, pessoas, banco de dados. Esta etapa é chamada de teste de sistema e inclui uma série de diferentes testes, cujo propósito primordial é pôr completamente à prova o sistema construído e deve verificar se todos os elementos do sistema foram adequadamente integrados e realizam as funções atribuídas. Segundo Beizer [4], alguns tipos de testes válidos são: teste de recuperação ou tolerância a falhas, teste de segurança ou invasão, teste de estresse e teste de desempenho. O teste de usabilidade pode ser considerado também como um tipo de teste de sistema.

A depuração não é teste mas sempre ocorre como uma conseqüência deste. Nesta etapa do ciclo de vida do teste (Figura 4.2) devem ser executados além de outros, o teste de regressão, que refaz todos os testes feitos anteriormente, para garantir que qualquer alteração no código não afetou nenhuma outra parte. Este teste é feito sempre que uma alteração no código for realizada durante a correção de um defeito ou de uma manutenção.

No teste de unidade são utilizados principalmente os critérios estruturais, uma vez que os requisitos de teste por eles exigidos limitam-se ao escopo da unidade. Estudos mostram [38] que são encontrados vários esforços de pesquisa no sentido de estender o uso de critérios estruturais para o teste de integração. Harrold e Soffa [17] apresentaram uma técnica para determinar as estruturas de definição-uso interprocedurais permitindo a aplicação dos critérios baseados em análise de fluxo de dados [14] em nível de integração. De forma semelhante, Linnenkugel e Müllerburg [33] propuseram uma série de critérios que estendem os critérios baseados em fluxo de controle e em fluxo de dados para o teste de integração. Haley e Zweben [16] propuseram um critério para selecionar caminhos em um módulo que deveria ser testado novamente na fase de integração com base em sua interface. Vilela, com base no conceito potencial uso, estende os critérios Potenciais Usos para o teste de integração [69].

Passos da atividade de teste

A atividade de teste deve executar alguns passos que são considerados como ciclo de vida do teste [52], descrito na Figura 4.2. Esses passos são essenciais a todas as estratégias de teste. Nessa figura é mostrado o fluxo de dados entre esses passos.

Duas classes de entrada são fornecidas ao processo de teste: Configuração de software (1): inclui uma especificação de requisitos de software, uma especificação de projeto e o código fonte; Configuração de teste (2): inclui um plano e procedimento de teste, ferramentas de teste que serão usadas e dados de teste (dados de entrada). Dados de teste mais resultados esperados são definidos como casos de teste. O processo de execução dos casos de teste gera os resultados dos testes (3).

Os resultados dos testes são confrontados com os resultados esperados (4) num processo chamado de Avaliação. Resultados esperados: são os resultados que o programa deveria obter se estivesse plenamente correto, esses dados são obtidos a partir de um oráculo, que pode ser o próprio testador.

Quando erros (5) são descobertos, infere-se a presença de defeitos e inicia-se a depuração. À medida que os resultados de teste são reunidos e avaliados uma indicação quantitativa da qualidade (6) e da confiabilidade (8) do software começa a vir à tona. Se defeitos graves,

que exijam modificação de projeto, forem encontrados com regularidade, a qualidade e a confiabilidade do software são suspeitas e testes adicionais são indicados. Se, por outro lado, as funções do software parecerem estar funcionando adequadamente e os defeitos encontrados puderem ser facilmente corrigidos (7), pode-se tirar uma entre duas conclusões: a qualidade e a confiabilidade do software são aceitáveis ou os testes são inadequados para revelar defeitos graves neste caso.

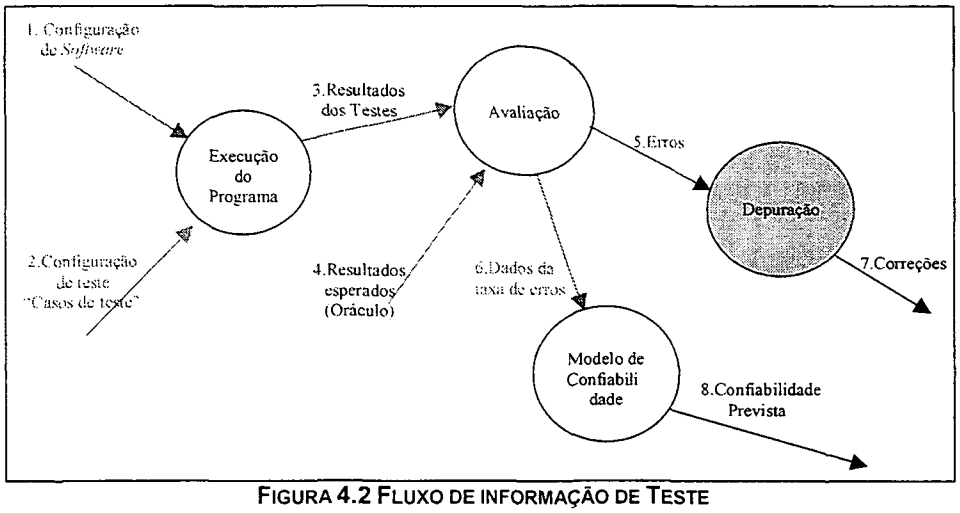


FIGURA 4.2 FLUXO DE INFORMAÇÃO DE TESTE

Percebe-se com esse fluxo, que a etapa de geração de casos de teste é bastante importante, pois dela dependerá a qualidade dos testes e os custos e esforços gastos nesta atividade.

4.2 ESTRATÉGIA PROPOSTA PELO *GARTNER GROUP*

A metodologia proposta pelo *Gartner Group* [10] considera que não é mais possível fazer testes apenas numa etapa de testes antes da implantação. A metodologia de desenvolvimento de aplicação atualmente mais utilizada é a abordagem espiral, portanto o ciclo de vida da atividade de teste também deve utilizar esta abordagem.

Aplicações cliente-servidor estão se tornando cada vez mais críticas no seu escopo e precisam ser amparadas por uma rigorosa metodologia de teste e ferramentas para que alcancem os níveis de qualidade e sucesso esperados. A maioria dos métodos de teste são caseiros-adaptados (*home-grown*) ou derivados de um esqueleto de uma metodologia tradicional de Administração de Dados - AD. A maioria dos vendedores de ferramentas de teste cliente-servidor fornecem alguma metodologia guia que é limitada e normalmente obrigam que sejam adquiridos serviços de consultoria e treinamento.

Segundo [10], grandes empresas e seus consultores propõem um modelo de estratégia para o teste de software cliente-servidor mostrado na Figura 4.3, recomendando que uma

metodologia de teste inclua no mínimo as seguintes fases:

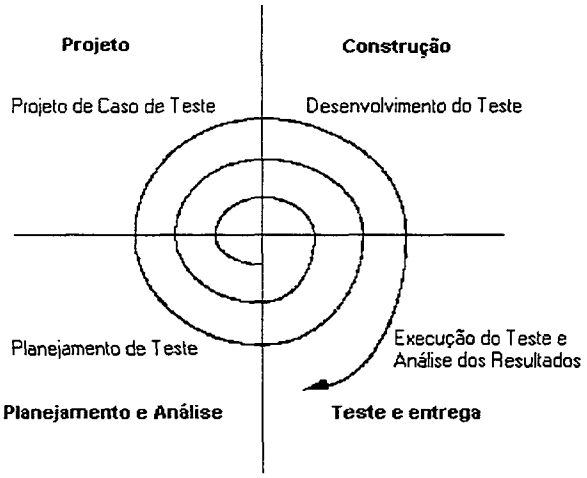


FIGURA 4.3 CICLO DE VIDA DO TESTE DURANTE O PROJETO

Planejamento do Teste - a primeira fase, onde deve ser produzido o plano de teste, que definirá os níveis e categorias de teste a serem conduzidos (isto é, funcional usuário-final, usabilidade, configuração e regressão). O plano também inclui uma lista de requisitos a serem testados ou verificados; critério de aceitação, aprovados pelo responsável do negócio (incluindo considerações de desempenho); regras e responsabilidades; ferramentas e técnicas a serem utilizadas e um cronograma para o teste. Conforme mostra a Figura 4.3 este planejamento deveria ser parte das especificações do sistema [41].

Projeto de Caso de Teste – esta segunda fase transforma em casos de teste documentados, os requisitos do sistema e comportamentos esperados pela aplicação, como especificados pelo responsável pelo negócio.

Desenvolvimento do teste – estes casos de teste devem ser transformados em programas, *scripts* que exercitarão o sistema em desenvolvimento, com a utilização de uma ferramenta própria. O mais comum é que uma ferramenta capture e grave como um *script* o que está sendo feito em tela, e a seqüência de entrada de dados e ações que estão sendo realizadas.

Execução e análise dos resultados – Muitos chamam só esta fase de teste. Aqui, a execução propriamente dos *scripts* vai gerar resultados para serem confrontados com os valores esperados e gerar novas "*baselines*" (linhas de base), que são um conjunto de documentos revisados e aprovados que representa um acordo para futuras evoluções de desenvolvimento e podem ser alterados somente através de um procedimento formal tal como gerência de mudanças e configuração [29].

4.3 PROCESSO PADRONIZADO TEST-RX

Este processo padronizado de teste de software é baseado na metodologia proprietária de teste de software oferecida pelas CSST Technologies[1] e pode ser usado em desenvolvimento de projetos de software cliente-servidor.

Avaliação e teste de software não são funções isoladas. Elas ocorrem em conjunto com outras atividades de desenvolvimento de software e dentro do contexto do ciclo de vida do mesmo. O Test-Rx [45] traça um conjunto de passos do processo e elementos de suporte que alcançarão os objetivos do teste do KPA [6], (Seção 3.1), estabelecendo os próprios níveis de comprometimento, fornecendo as habilidades requeridas e definindo as atividades de teste e evolução necessárias.

Test-Rx é um processo que inclui elementos de suporte para uso em qualquer projeto de teste de desenvolvimento de software. O propósito do processo é fornecer uma linha base para as atividades de teste de software que fazem parte do projeto. O processo consiste de uma série de passos. Cada passo consiste de um conjunto de tarefas. É possível produzir um produto parcial que pode ser entregue. Conceitualmente o processo poderia ser considerado como um modelo espiral em que tarefas de diferentes passos podem ser feitas em paralelo e, é cíclico, pode-se voltar ao passo anterior, atualizar alguns dados, refazer as tarefas e continuar. A Tabela 4.1 e a Tabela 4.2 ilustram e resumem este processo.

TABELA 4.1 ATIVIDADES DO PROCESSO DE TEST-RX

Atividades do Processo de Teste	Gerenciamento de configuração	Rastreamento de defeito	Automação de teste	Revisões pessoais	Métricas de teste
Montar uma equipe de teste	Não	Não	Não	Requer	Não
Executar análise de risco	Opcional	Não	Opcional	Requer	Requer
Definir requisitos/objetivos de teste	Opcional	Não	Requer	Requer	Requer
Desenvolver planos de teste	Opcional	Não	Requer	Requer	Requer
Projetar casos de teste	Opcional	Não	Requer	Requer	Não
Executar testes de integração e unidade	Requer	Requer	Requer	Requer	Requer
Executar testes de sistema	Requer	Requer	Requer	Requer	Requer
Analisar e listar os resultados do teste	Requer	Não	Opcional	Requer	Requer
Executar teste de regressão	Requer	Requer	Requer	Requer	Requer
Analisar e listar os resultados do teste de regressão	Requer	Requer	Requer	Requer	Requer

A Tabela 4.1 apresenta as atividades pertinentes ao processo de *Test-Rx* juntamente com os elementos que dão suporte a estas atividades, que são Gerenciamento de configuração, Rastreamento de defeito, Automação de teste, Revisões pessoais e Métricas de teste. Para

[1] CSST Technologies, Inc., and SQM Solutions são divisões da Toro Technologies, Inc.

cada um destes elementos a tabela mostra quais atividades são requeridas, quais são opcionais ou quais não precisam ser executadas.

TABELA 4.2 PASSOS DO PROCESSO TEST-RX DENTRO DO CICLO DE VIDA

Processo de Teste	*Estágio do ciclo de vida*	*Pessoal envolvido*
1. Montar uma equipe de teste	Análise	Gerente de Projeto, Gerente de teste
2. Executar análise de risco	Análise	Gerente de Projeto, Gerente de teste
3. Definir requisitos/objetivos de teste	Análise	Gerente de teste, Analistas de teste
4. Desenvolver planos de teste	Análise/Projeto	Analistas de teste
5. Projetar casos de teste	Análise/Projeto	Analistas de teste
6. Executar testes de integração e unidade	Construção	Desenvolvedores (unidade), Equipe (integração)
7. Executar testes de sistema	Teste	Equipe de teste
8. Analisar e listar os resultados do teste	Teste	Gerente de teste, Analistas de teste
9. Executar teste de regressão	Manutenção/Produção	Analistas de teste
10. Analisar os resultados teste de regressão	Manutenção/Produção	Analistas de teste

Para executar o passo 2, que corresponde à análise de risco, é usada a Tabela 4.3 como balizadora.

TABELA 4.3 CLASSIFICAÇÃO DOS DEFEITOS UTILIZADA NO TEST-RX

Grau	Descrição	Ação
1. Crítico	Os defeitos resultam em falhas do sistema ou de parte dele e não tem como fazer de outra forma ou seja o trabalho pára completamente.	Resolver imediatamente
2. Prioritário	Os defeitos resultam em falhas do sistema, entretanto existem alternativas de processo (manualmente, por exemplo) que produzirão os resultados desejados	Dar atenção alta
3. Regular	Os defeitos não resultam em falhas mas produzem resultados inconsistentes, incompletos ou incorretos ou prejudicam a usabilidade do mesmo	Fila normal média prioridade
4. Pouca importância	Os defeitos não causam uma falha, não prejudicam a usabilidade e os resultados do processamento desejado são obtidos contornando o defeito	Baixa Prioridade
5. Exceção de qualidade	O defeito resulta de uma requisição de alteração ou melhoria, podem ser deferidos ou ignorados	Deferir ou não, (longo prazo)

4.4 TESTES PROPOSTOS POR MOSLEY [45]

Existe um consenso na literatura, tendo-se consciência de que em uma arquitetura cliente-servidor tem que ser levada em consideração cada camada (de apresentação, dados e aplicação) isoladamente. Há tipos de testes específicos para cada camada.

Camada de apresentação (cliente)

Conforme Mosley [45] descreve, para a interface GUI podem ser realizados:

1. testes de usabilidade – este teste é uma parte crítica da camada de apresentação. Devem ser verificadas as mensagens para o usuário, facilidade de uso dos componentes, localização dos objetos na tela, cores ou sons impróprios, etc. Este tipo de teste pode contribuir para o aprendizado do sistema e diminuir os gastos com treinamento. Uma lista de verificação para este tipo de teste é disponibilizada em [44];

2. teste de intuitividade de ícones –um grupo de pessoas usadas para este teste recebe os ícones do sistema sem o descritivo deles e essas pessoas são questionadas sobre o que aquele ícone representa;
3. teste de distribuição de cartas-para-ícones – são usadas cartas com os comandos e ícones do sistema impressos e as pessoas têm que relacionar os nomes dos comandos aos ícones;
4. ensaio-geral do modelo da tela-principal do sistema – usa um modelo em papel da tela principal do sistema e as pessoas usadas para este tipo de teste são indagadas sobre o que elas imaginam que possam acessar através de cada ícone.

Camada de negócio e camada de dados (servidor)

Na camada do meio, deve-se testar a conexão com a base de dados, usando aplicativos do próprio banco. Por exemplo, devem ser checados: se os componentes estão conseguindo ser instanciados no MTS (*Microsoft Transaction Server,* monitor de transação da Microsoft); se a rede está proporcionando o devido acesso aos diretórios; se toda a infra-estrutura para esta arquitetura rodar está funcionando com cada componente no seu papel.

A camada de negócio, que em termos lógicos também é considerada a camada do meio, corresponde aos componentes que rodam toda a lógica de negócio da aplicação. Nesta camada são realizados os mesmos testes identificados para a camada do servidor.

1. teste de carga – testes com um único usuário acessando uma função e várias funções e, posteriormente simular centenas de usuários acessando uma única função e várias funções simultaneamente;
2. teste de volume – procurar encontrar o ponto fraco, ou capacidade máxima de manipulação de dados, durante um período extenso;
3. teste de estresse – este teste procura verificar se o sistema tem capacidade de manipular um grande número de transações em um período de pico.
4. teste de desempenho (*performance)* – este teste verifica o tempo de resposta geralmente associados com os teste de carga e estresse;
5. teste de recuperação ou devolução de dados – dados podem ser perdidos quando hardware falha ou quando arquivos são apagados voluntária e involuntariamente, ataque de vírus, desastres naturais, falhas provocadas pelo conflito de memória do sistema operacional, quando aponta para um caminho que não existe, etc.;

6. teste de cópia de segurança e restauração dos dados - planejamento da estratégia de *backup* e verificação das cópias;
7. teste de segurança dos dados – não só uma validação de quem pode acessar o que, como também verificar se ficam armazenadas no código os *logins* de acesso à base de dados, de que forma são feitos os acessos, quem autentica se o usuário é quem diz ser;
8. teste de integridade dos dados – a) garantir a atomicidade: verificar se são feitos controles de transação; b)consistência: se dados são alterados corretamente; c) isolamento: se são feitos os bloqueios para que não haja dois usuários alterando o mesmo dado; e d) durabilidade: garantir que os dados alterados superam a falhas do sistema.

4.5 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE UNIFICADO

No final dos anos 90, três autores consagrados na área de orientação a objetos (Jacobson, Booch e Rumbaugh) reuniram-se para desenvolver o *Unified* Software *Development Process* (UP) [29]. Posteriormente, o UP foi instanciado e especificado pela Rational Software Corporation [57], resultando no *Rational Unified Process* (RUP), descrito por Kruchten em [32]. Conforme relatado no artigo [34], a característica fundamental é ser baseado em componentes, ou seja, o software desenvolvido é formado por componentes de software que se comunicam através de interfaces bem definidas. O padrão adotado para representação dos modelos é a *Unified Modeling Language* (UML).

O RUP foi concebido tomando como base três conceitos fundamentais: dirigido por *use case* (caso de uso), centrado na arquitetura, iterativo e incremental, conforme descrito a seguir.

- **Dirigido por caso de uso:** "um caso de uso (*use case*) é uma parte da funcionalidade do sistema que fornece ao usuário um resultado de valor" [29]. Os casos de uso são utilizados para a capturar e definir os requisitos funcionais do sistema de software, facilitando a comunicação e o entendimento entre analistas e usuários finais. São também utilizados para projetar os casos de teste.
- **Centrado em arquitetura:** no UP os casos de usos e a arquitetura vão sendo desenvolvidos em paralelo. O conceito de arquitetura engloba os aspectos mais relevantes, tanto estáticos, como dinâmicos, do sistema.
- **Iterativo e Incremental:** o UP utiliza pequenos ciclos de projeto (mini-projetos) que correspondem à uma iteração e que resultam em um incremento no software. Iterações

referem-se a passos no fluxo de trabalho (*workflow*) e incrementos referem-se às evoluções do produto.

O desenvolvimento de software efetuado através do RUP é incremental e cada incremento é desenvolvido utilizando-se quatro fases: iniciação, elaboração, construção e transição. [29],[59], [34]. Estas fases compõem o ciclo de desenvolvimento. Após a fase de transição, o produto pode voltar a percorrer cada uma das fases, constituindo a fase de evolução, na qual se produz uma nova geração do produto [59]. Estas fases com seus respectivos processos bem como a execução de cada um deles durante as fases estão bem representados através da Figura 4.4 [29], [34] [59].

- **Iniciação:** fase de compreensão do problema e da tecnologia através da definição dos casos de uso mais críticos. No final desta fase deve-se ter definido o escopo do produto, os riscos e ter demonstrado que o projeto é viável do ponto de vista do negócio da organização.

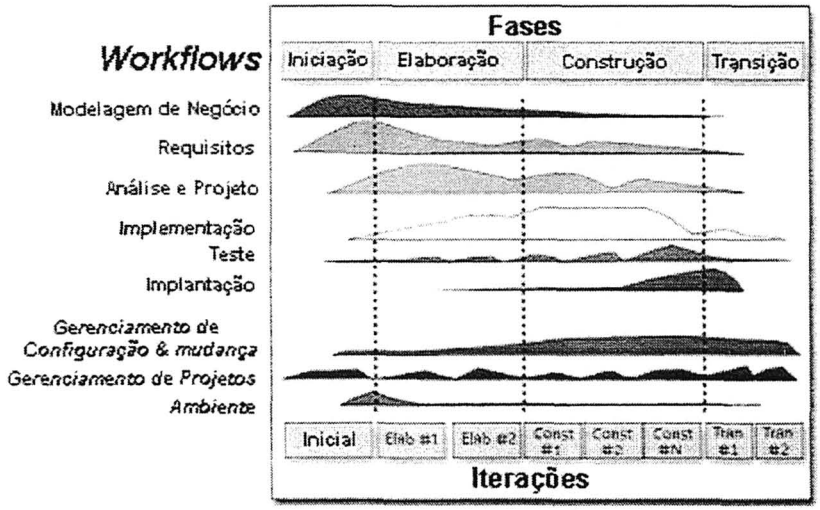


FIGURA 4.4 EXECUÇÃO DE WORKFLOWS DENTRO DE UM CICLO DE VIDA ITERATIVO

- **Elaboração:** fase de descrição da arquitetura do software na qual os requisitos que mais impactam na arquitetura são capturados em forma de casos de uso. No final da fase de elaboração deve ser possível estimar custos, elaborar o cronograma e o plano de construção.
- **Construção:** fase na qual o software é construído e preparado para a transição para os usuários. Além do código, propriamente dito, também são produzidos os casos de teste e a documentação.
- **Transição:** fase de treinamento dos usuários e transição do produto para utilização.

Cada uma das quatro fases do RUP é adicionalmente dividida em iterações e finalizada com um ponto de checagem que verifica se os objetivos daquela fase foram alcançados. Toda iteração é organizada em termos de *workflows* de processo, que são conjuntos de atividades realizadas por responsáveis, conforme ilustrado na Figura 4.4.

Os principais *workflows* do processo são descritos a seguir: [34], [59]

1) Modelagem de Negócio: objetiva um entendimento comum através dos casos de uso de negócio;

2) Requisitos: objetiva capturar os requisitos que serão atendidos pelo produto de software;

3) Análise e Projeto: objetiva compreender mais precisamente os casos de usos definidos no fluxo de trabalho de requisitos, produzindo um modelo já voltado para a implementação que deverá estar adequado ao ambiente de implementação;

4) Implementação: objetiva a organização do código em termos de implementação de subsistemas, classes e objetos em termos de componentes, testando-os isoladamente e integrando-os aos códigos produzidos;

5) Teste: objetiva analisar, através de testes, se os requisitos foram atendidos e que os defeitos serão removidos antes da implantação; e

6) Entrega: objetiva produzir *releases* do produto e entregá-los aos usuários finais. Isto pode incluir atividades de beta-teste, migração de dados ou software existente e aceitação formal.

As formas de executar os testes dependem de vários fatores: domínio da aplicação, orçamento, política da empresa, tolerância a riscos e pessoal. Quanto será investido em teste depende diretamente de como é avaliada a qualidade e tolerância a risco em uma empresa.

4.5.1 Ferramenta de Teste da Rational

Para que se possa compreender melhor a estratégia de teste associada à ferramenta de teste da Rational [57], a qual implementa a estratégia UP, pode ser visualizado na Figura 4.5 o ciclo de vida do teste sem considerar as fases do projeto. Esta figura apresenta como as diferentes atividades estão interconectadas olhando apenas uma iteração.

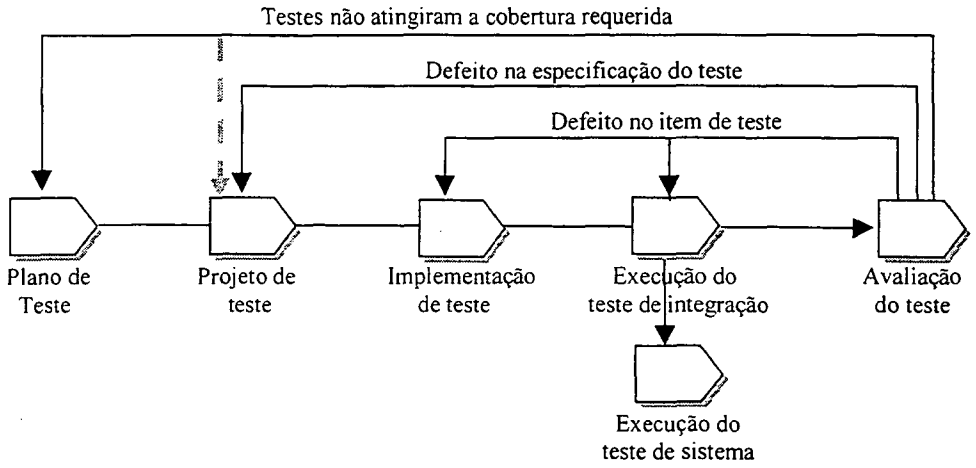


FIGURA 4.5 CICLO DE VIDA DO TESTE DENTRO DO PROJETO

No ciclo de vida de desenvolvimento do sistema, o sistema é refinado através de iterações. Acréscimos e refinamentos que são feitos para o teste são executados para cada etapa, acumulando um corpo de teste, que são usados para o teste de regressão em estágios futuros. Esta abordagem implica que o teste tem que ser feito durante todo o processo. A abordagem iterativa dá um grande foco para o teste de regressão, pois a maior parte dos dados usados na iteração x serão usados como dados de regressão na iteração x + 1.

A metodologia proposta pela ferramenta descreve cada fase do ciclo de vida do teste, mas destaca como mais importante o Plano de Teste. Este documento deve conter:

- **Propósito** - contém informações sobre o propósito e objetivos do teste dentro do projeto. Além disso, identifica as estratégias a serem utilizadas para implementar e executar os testes e os recursos necessários.
- **Identificar os requisitos para os testes** – os requisitos devem ter um comportamento mensurável e observável. Não há um relacionamento de um para um dos requisitos com caso de uso. Cada caso de uso pode ter um ou mais requisitos para testar.
- **Avaliação de risco para estabelecer as prioridades de teste** – estabelecer as prioridades é importante para garantir que os esforços dos testes estão focados nos requisitos mais apropriados, garantir que os requisitos que possuem mais riscos e que são mais críticos e significativos, sejam verificados primeiro.

  Para avaliar os riscos e estabelecer as prioridades, são avaliados 3 itens, segundo a Tabela 4.4., Efeitos, causa e probabilidade do caso de uso falhar. Considerar o valor mais alto dos 3 itens.

  Depois se determina o perfil operacional, classificando Alto quando o caso de uso é usado com muita freqüência ou por muitos atores; Médio se usado freqüentemente por

vários atores; e Baixo se não usado com freqüência e por poucos atores. Deve ser levado em consideração o número de vezes que um ator executa o caso de uso e o número de atores que executam o caso de uso.

TABELA 4.4 CLASSIFICAÇÃO DOS RISCOS

Classificação	Características
Alta	Alto risco, defeito não pode ser tolerado. Provoca uma exposição externa forte. A empresa sofrerá grandes perdas financeiras, endividamento ou uma perda irrecuperável da reputação.
Média	Risco médio, tolerável mas não desejável. Exposição externa mínima. A empresa pode sofrer financeiramente, mas há um endividamento ou perda da reputação sob controle.
Baixa	Baixo risco, tolerável. Pouca ou nenhuma exposição externa, a empresa pode ter pouca ou nenhuma perda financeira. A reputação da empresa não é afetada.

Depois deve ser analisado o usuário quanto a sua experiência e tolerância a falhas, e também com relação a possibilidade de entregar o sistema sem o caso de uso (obrigações contratuais).

O estabelecimento da prioridade deve ser calculado multiplicando os valores Alto por 5, Médio por 3 e Baixo por 1. Observar a somatória dos valores e atribuir valor final Alto para >30, Médio para valores entre 20 e 30 inclusive, e Baixo para valores menores que 20. Deve ser definida a magnitude que tem este valor como por exemplo, Alto é obrigatório o teste, Médio deve testado após todos os de valor final Alto e Baixo, pode ser testado mas não antes que todos os de valores Alto e Médio.

- **Estratégias de teste** – estabelecem onde são especificados os tipos de testes a serem implementados e seus objetivos, estágio em que o teste será implementado, técnicas utilizadas, medidas e critérios usados para avaliar o término e os resultados do teste e algumas considerações especiais que podem afetar o esforço de teste.

4.6 CONSIDERAÇÕES FINAIS

Este capítulo descreve os principais trabalhos encontrados na literatura e que serviram como base para propor a estratégia ETACS, descrita no Capítulo 6.

Há diversos fornecedores investindo esforços na área de teste, principalmente porque agrega qualidade ao produto. Teste é uma atividade que demanda tempo e organização. Algumas empresas já formalizaram seu processo para execução da atividade de teste e alguns desses trabalhos foram abordados neste capítulo. Uma combinação de características desses trabalhos com o foco na arquitetura cliente-servidor formaram a base para a elaboração da ETACS.

O estabelecimento de prioridade proposto na ferramenta da Rational é bastante interessante e completo e foi aplicado na ETACS e alterados alguns itens pela dificuldade na aplicação. Os testes propostos por Mosley foram integralmente aproveitados acrescentando apenas algumas orientações que foram considerados pertinentes.

O *Gartner Group* considera em uma das fases do ciclo de vida do desenvolvimento com teste a inclusão de uma etapa para geração de *scripts* dos casos de testes projetados. Geração de *scripts* pressupõe utilização de ferramenta e a ETACS não obriga a utilização, apenas aconselha o uso de uma ferramenta para a execução dos testes que se fizerem mais necessários.

Pressman trabalha um pouco mais no ciclo de vida evolutivo de desenvolvimento mas definiu níveis de teste que podem ser adequados a qualquer ciclo e a ETACS procurou estabelecer estes níveis de teste para o ciclo iterativo apresentado pelo RUP, no qual a ETACS foi baseada, considerando também o ambiente cliente-servidor 3 camadas.

Como pode ser observado, todos os trabalhos acabam focalizando de uma maneira um pouco diferente os mesmos itens. Porque realmente eles são os mais importantes. Alguns fornecem mais detalhes de como fazer as coisas, outros não. A ETACS procurou se apoiar nestas metodologias, processos e estratégias com o foco no ambiente cliente-servidor.

5 FUNDAMENTOS EMPÍRICOS

Além dos trabalhos descritos no capítulo anterior, a ETACS também leva em consideração um estudo empírico realizado através da análise de informações obtidas por meio de uma pesquisa realizada em uma empresa que utiliza tecnologia cliente-servidor.

Com a finalidade de levantar a situação da atividade de teste realizada nos sistemas dentro desta empresa foi elaborado um questionário e distribuído para 138 pessoas, das quais 48 pessoas responderam, o que representa 35% das pessoas que foram questionadas. Conforme pode ser observado na Figura 5.1, dessas 48 pessoas, 8 são gerentes e coordenadores, 34 analistas e líderes de projeto e 6 programadores e técnicos de suporte. O objetivo é verificar a eficiência e os problemas existentes para a realização do teste. As respostas coletadas foram analisadas estatisticamente e os principais resultados que serviram como base para a ETACS são apresentados a seguir.

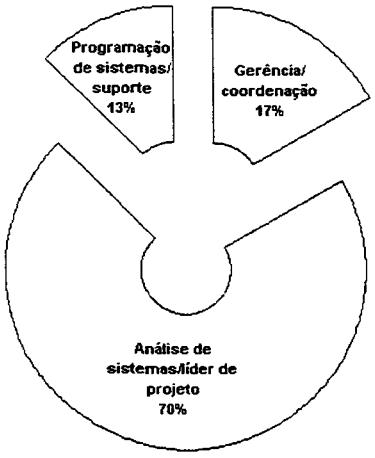


FIGURA 5.1 PERFIL DAS PESSOAS QUE RESPONDERAM

Os dados obtidos do questionário com as perguntas, objetivos e respostas tabuladas estão disponíveis no APÊNDICE B.

5.1 ANÁLISE DOS RESULTADOS

A princípio foi elaborado um questionário e validado com 5 pessoas da área de informática, para posteriormente ser enviado aos técnicos. O questionário foi divulgado através do *Lotus Notes* por ser uma ferramenta bem disseminada na empresa e possibilitar uma tabulação bruta dos dados. O questionário contém 5 seções: Perfil do Testador, Importância do Teste, Execução dos Testes, Verificando Critérios e Técnicas e Identificando maiores dificuldades.

Com base nas 48 respostas obtidas foi possível fazer algumas considerações que são importantes para auxiliar o estabelecimento de uma estratégia de teste. Para fazer o cruzamento das respostas e alguns gráficos foi utilizado o software STATISTICA for Windows [64] e o Microsoft Excel.

1ª análise – Forma e motivação dos testes

Como pode ser observado na Figura 5.2, 44% das pessoas testam os sistemas com a motivação de comprovar que seu programa está correto.

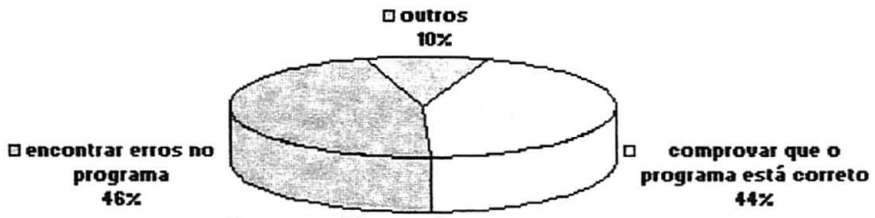


FIGURA 5.2 MOTIVAÇÃO DOS TESTADORES

Pessoas que têm esse objetivo em mente tentam proteger seus sistemas, não querem que ele apresente muitos erros e a tendência natural é que essas pessoas entrem com os dados que deveriam dar certo e acabam não testando as situações em que provavelmente vão ocorrer erros.

2ª análise – Execução de Teste de desempenho

Como pode ser observado pela Figura 5.3, 83% (40 pessoas) nunca ou somente às vezes fazem teste de desempenho.

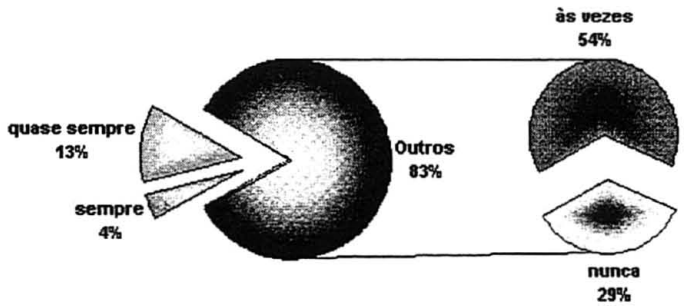


FIGURA 5.3 REALIZAM TESTE DE DESEMPENHO NOS SISTEMAS

Considerando somente estas 40 pessoas (83%), foi verificado quais delas encontram problemas de desempenho em seus sistemas e também as que têm dificuldade de simular o ambiente do cliente. A Tabela 5.1 apresenta este cruzamento. A coluna E1B-Performance tabula as pessoas com problema de desempenho e as colunas E4B-Simular as que têm dificuldade de simular o ambiente com número de usuários e carga de dados. Estes dados estão representados graficamente na Figura 5.4.

TABELA 5.1 ENCONTRAM PROBLEMA DE DESEMPENHO X DIFICULDADE DE SIMULAR O AMBIENTE

E1B-Perfomance	E4B-Simular Não	E4B-Simular Sim	Total
Não	4	19	23
Não %	10,00%	47,50%	57,50%
Sim	4	13	17
Sim %	10,00%	32,50%	42,50%
Total	8	32	40
Total %	20,00%	80,00%	

Conforme pode ser observado, das pessoas que raramente fazem teste de desempenho, 17 encontram problemas de desempenho, o que representa 42,5% e 13 dessas 17 pessoas, atribuem como principal problema para realizar esta tarefa, a dificuldade de simular o ambiente do cliente com carga de dados e número de usuários. Um outro detalhe a ser observado na Tabela 5.1 é que 80% das pessoas que nunca ou, às vezes, fazem o teste de desempenho, consideram simular o ambiente do cliente com carga de dados e número de usuário como um problema na atividade de teste.

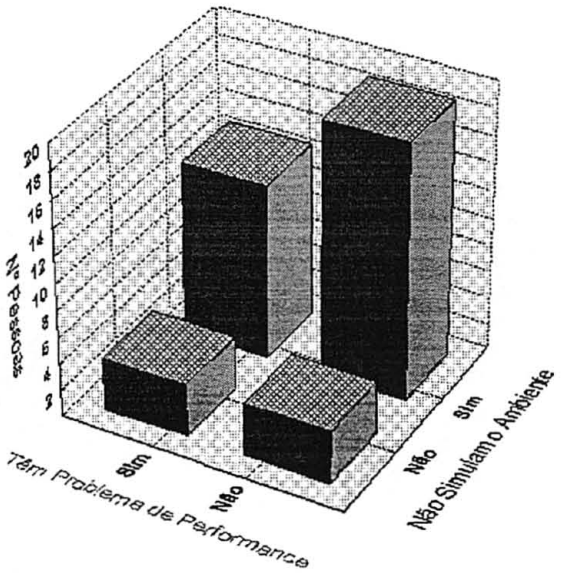


FIGURA 5.4 ENCONTRAM PROBLEMA DE DESEMPENHO X NÃO CONSEGUEM SIMULAR O AMBIENTE

Com base nestes dados pode-se concluir que simular o ambiente do cliente com carga de dados e número de usuários é uma dificuldade que se solucionada deveria resolver uma grande parte dos problemas de desempenho e deve ser uma preocupação da estratégia.

3ª análise - Registros

Pode ser observado na Tabela 5.2 e Figura 5.5 que a maioria absoluta não registra o tempo gasto nem os recursos utilizados para calcular o custo da atividade de teste.

TABELA 5.2 REGISTRO DOS GASTOS PARA CALCULAR O CUSTO DA ATIVIDADE DE TESTE

Respostas	Total	Freqüência
sempre	5	10,4%
quase sempre	4	8,3%
às vezes	14	29,2%
nunca	22	45,8%
Branco	3	6,3%
Total	48	100,0%

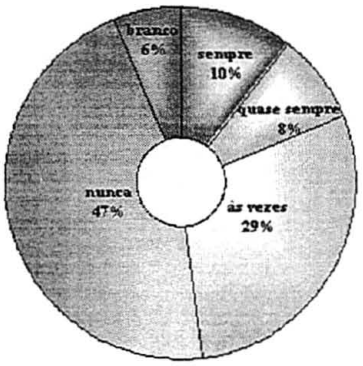


FIGURA 5.5 REGISTRAM TEMPO GASTO COM TESTE

Adicionando-se o fato que pode ser observado na Tabela 5.3, dificilmente as pessoas registram os erros encontrados e corrigidos durante a atividade de teste. Fica complicado manter um histórico para verificar os benefícios oriundos desta atividade e com isso justificar junto ao cliente a importância desta atividade e prorrogação do cronograma.

TABELA 5.3 REGISTRO DOS ERROS ENCONTRADOS DURANTE OS TESTES

Respostas	Total	Freqüência
sempre	6	12,5%
quase sempre	3	6,3%
às vezes	18	37,5%
nunca	21	43,8%
Branco	0	0,0%
Total	48	100,0%

4ª análise – Executam Teste de regressão

Pode ser observado através da Tabela 5.4, que os poucos que mantêm uma lista dos dados de teste não costumam atualizá-los.

TABELA 5.4 MANTÉM LISTA DO QUE É TESTADO-C3 X C9-ATUALIZA DADOS DE TESTE

C3-Mantém	C9 Branco	C9 Sempre	C9 Q.Sempre	C9 Às vezes	C9 Nunca	Totais
Branco	0	0	0	2	0	2
Sempre	0	0	4	1	1	6
Q.Sempre	0	1	4	5	4	14
Às vezes	1	1	1	8	8	19
Nunca	0	0	0	2	5	7
Totais	1	2	9	18	18	48

Se a lista do que é testado não for atualizada quando os dados forem modificados, ela não serve para ser usada nos testes de regressão.

Para verificar como as pessoas atualizam os dados de teste foi realizado um cruzamento de questões. Considerando somente aquelas pessoas que Sempre ou Quase Sempre mantêm uma lista do que é testado. O resultado pode ser visto na Tabela 5.5

TABELA 5.5 ATUALIZA DADOS DE TESTE (DOS QUE QUASE SEMPRE OU SEMPRE MANTÊM LISTA)

C3	C9 Sempre	C9 Q.Sempre	C9 Às vezes	C9 Nunca	Row Totals
Sempre	0	4	1	1	6
Q.Sempre	1	4	5	4	14
Total	1	8	6	5	20

Dos 20 que Quase Sempre e Sempre mantêm lista do que é testado, verifica-se na Tabela 5.5 e Figura 5.6 que 6 às vezes atualizam os dados quando estes são modificados e 5 nunca o fazem. Nesses casos, a lista serve como uma documentação mas não para os testes de regressão.

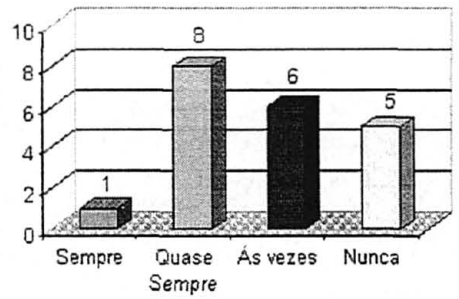


FIGURA 5.6 ATUALIZAM DADOS DE TESTE (DOS QUE SEMPRE/Q.SEMPRE MANTÊM LISTA)

Das pessoas que responderam ao questionário, pode ser observado através da Figura 5.7 que a maioria (54,2%), às vezes ou nunca faz o teste de regressão.

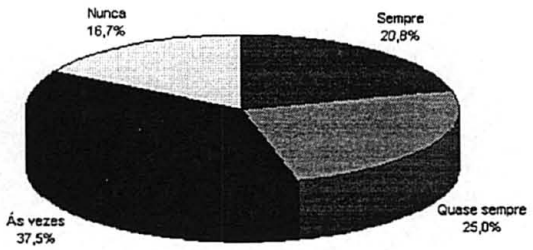


FIGURA 5.7 FAZ TESTE DE REGRESSÃO

Teste de regressão tem como objetivo, depois que um programa é alterado, refazer todos os testes feitos anteriormente para garantir que a alteração no programa não provocou nenhum impacto em outro lugar do sistema.

Pode-se observar através da Figura 5.8, o que é esperado, dos 20 entrevistados que mantêm uma lista do que é testado, 8 (que equivale a 40%) sempre ou 5 (que equivale a 25%) quase sempre fazem teste de regressão.

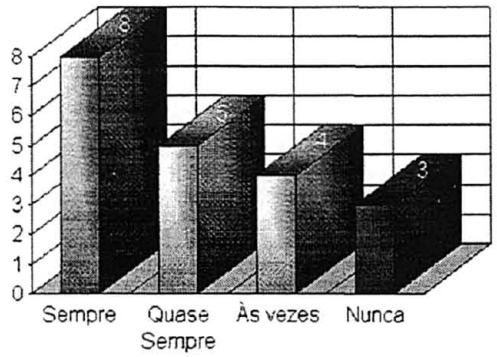


FIGURA 5.8 FAZEM TESTE DE REGRESSÃO (DOS QUE SEMPRE/QUASE SEMPRE MANTÉM LISTA)

Verifica-se na Figura 5.9 que, quem Às vezes ou Nunca mantém uma lista do que é testado (no total 26 pessoas), 13 às vezes (que corresponde a 50%) e 5 nunca (que corresponde a 19,2%) fazem teste de regressão. Por isso, é necessário aumentar o número de pessoas que guardam esta lista para poder facilitar o teste de regressão e também o número de pessoas que realizam esse teste. Uma vez que relembrar todos os testes de cabeça, é difícil e arriscado.

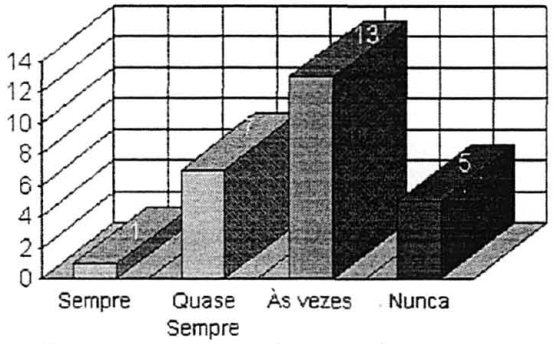


FIGURA 5.9 FAZEM TESTE DE REGRESSÃO (DOS QUE ÀS VEZES/NUNCA MANTÊM LISTA)

5ª Análise – Geração dos dados de teste

Pode-se verificar, pela Figura 5.10, que 43,8% Quase Sempre (corresponde a 21 dos entrevistados) e 27,1% Sempre (corresponde a 13 dos entrevistados) geram os dados de teste para a realização dos mesmos a partir dos requisitos do usuário.

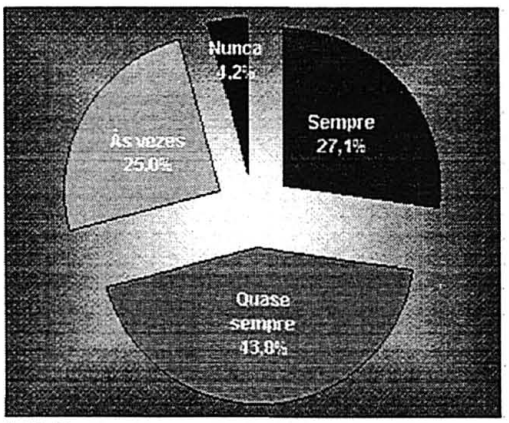


FIGURA 5.10 GERAM DADOS A PARTIR DOS REQUISITOS DO USUÁRIO

Pode ser constatado através da Tabela 5.6 e da Figura 5.11, que 60,4% (17 que corresponde 35,4% e 12 a 25%) Às vezes ou Nunca geram dados de teste a partir do código do programa (para fazer testes estruturais).

TABELA 5.6 GERAM DADOS A PARTIR DO CÓDIGO DO PROGRAMA

	Count	Percent
Branco	2	4,2
Sempre	8	16,7
Quase Sempre	9	18,8
Às vezes	17	35,4
Nunca	12	25,0
Total	48	100,0

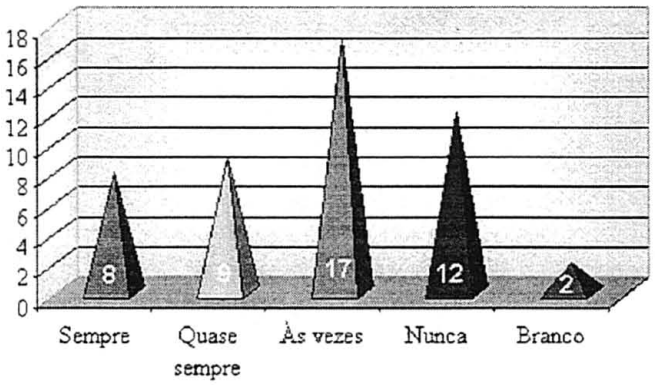


FIGURA 5.11 GERAM DADOS A PARTIR DO CÓDIGO DO PROGRAMA

Também pode ser verificado pela Figura 5.12 que 64,6% (31,3% corresponde a 15 pessoas + 33,3% que corresponde a 16 pessoas) Quase Sempre ou Sempre geram os dados sem aplicar algum critério formal, a partir de sua experiência do que o usuário poderia digitar de errado.

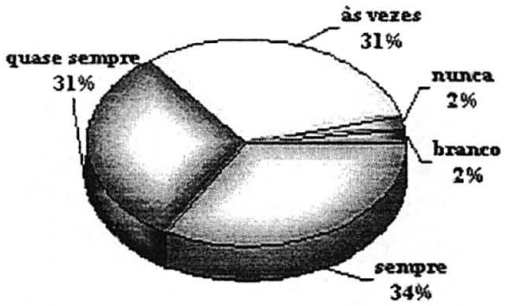


FIGURA 5.12 GERAM DADOS SEM CRITÉRIO FORMAL (ALEATORIAMENTE)

Com tudo isso, pode-se concluir que são feitos mais testes funcionais do que estruturais na empresa e muitos entrevistados geram dados sem aplicar nenhum critério formal. Uma das conseqüências de não se fazer testes estruturais é a produção de muitos erros de código, isto é verificado através da Figura 5.13, que mostra que 56,3% dos entrevistados (corresponde a 27) apontam erros de código como um dos problemas mais encontrados nos sistemas enviados para os clientes.

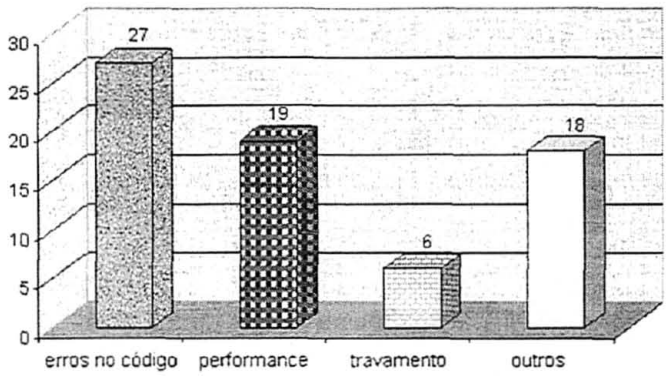


FIGURA 5.13 PROBLEMAS MAIS ENCONTRADO NOS SISTEMAS

6ª Análise – Realização de testes

Na Figura 5.14 é constatado que 42% dos entrevistados Sempre ou Quase Sempre fazem alguns testes iniciais no sistema e, posteriormente, enviam-no para o cliente testar. Essa fase muitas vezes é considerada como fase de testes, mas na realidade é o teste beta.

FIGURA 5.14 FAZEM TESTE BETA

Por outro lado o teste alfa, que é trazer o cliente para o ambiente de desenvolvimento no intuito de fazer os testes em um ambiente controlado, segundo o resultado da pesquisa, às vezes (38%) ou nunca (35%) é feito. Visto na Figura 5.15.

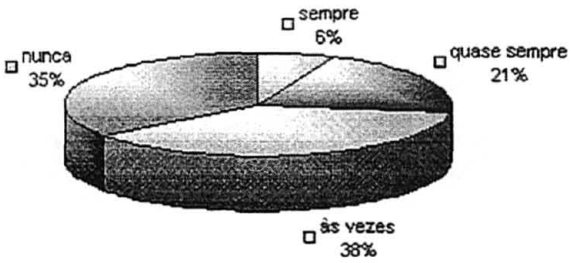


FIGURA 5.15 REALIZAM TESTE ALFA

Teste de integração que consiste em integrar partes do código ou funções, são realizados pela grande maioria das pessoas (39,6% Sempre e 41,7% Quase Sempre), verificados pela Figura 5.16.

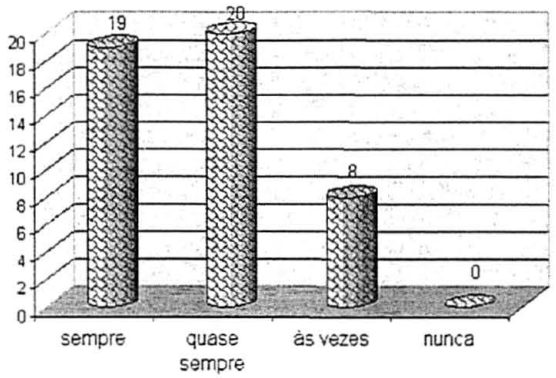


FIGURA 5.16 FAZEM TESTE DE INTEGRAÇÃO

Os testes de unidade, testes realizados sobre uma parte menor do programa e feito isoladamente, utilizando o depurador das linguagens para analisar os conteúdos das variáveis em tempo de execução, são utilizados às vezes pelas pessoas (56,3% das pessoas correspondendo a 27), conforme visto na Figura 5.17.

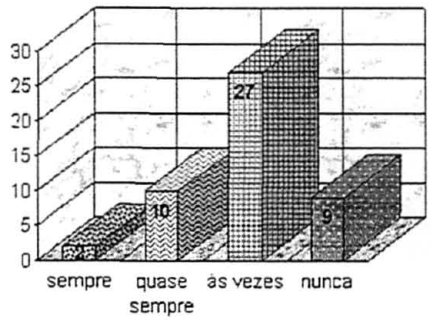


FIGURA 5.17 TESTE DE UNIDADE FEITO COM RECURSOS DO DEPURADOR

O depurador que é um recurso das linguagens de desenvolvimento não tem sido utilizado como ferramenta para auxiliar no teste de unidade.

7ª Análise – Dificuldades

Conforme a Tabela 5.7 e Figura 5.18, a dificuldade mais registrada para a realização dos testes foi a não possibilidade de simulação do ambiente do cliente com número de usuários e carga de dados, este item foi também tratado anteriormente na 2ª análise.

TABELA 5.7 DIFICULDADES ENCONTRADAS PARA A REALIZAÇÃO DOS TESTES

Respostas	Total	Freqüência
falta de um roteiro	20	41,7%
você não consegue simular o ambiente do cliente com número de usuários e carga de dados	36	75,0%
você não lembra todas as entradas para verificar todas as situações possíveis	8	16,7%
você não planejou um teste com valores de entrada e resultados que deveriam ocorrer	9	18,8%
outros	9	18,8%
Branco	0	0,0%
Total	48	

Mas outra dificuldade para a realização dos testes, apontada por 41,7% dos entrevistados, foi a falta de um roteiro, também verificado na Tabela 5.7 e na Figura 5.18.

FIGURA 5.18 MAIORES DIFICULDADES PARA REALIZAR OS TESTES

Questionados sobre o que poderia melhorar a atividade de teste, 52,1% dos entrevistados, conforme Figura 5.19, acreditam que roteiros poderiam melhorar a atividade de teste, 75% acredita que a utilização de uma ferramenta vai minimizar as dificuldades para a realização dos testes, e vai melhorar a qualidade, tanto dos testes como do produto final.

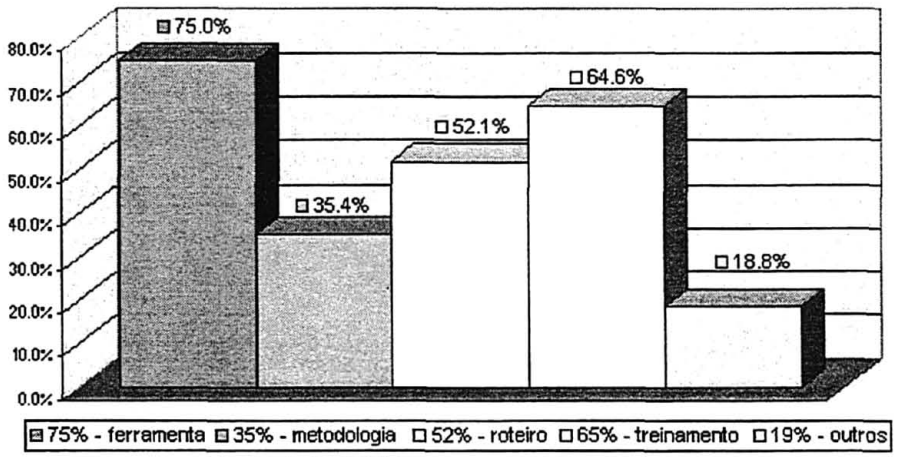


FIGURA 5.19 AUXÍLIO PARA A ATIVIDADE DE TESTE

8ª Análise – Como são feitos os testes

Os dados apresentados na Tabela 5.8 não evidenciam muito a teoria, que quanto mais testado um programa, menos manutenção será necessária. Talvez porque os técnicos que fazem a manutenção não sejam os mesmos que desenvolvem e a pergunta não foi dirigida à análise de um sistema e sim às atividades que os técnicos executam como um todo.

Outro detalhe que pode ser observado nesta tabela é que mais de 60% do pessoal gasta de 4 a 16 horas com testes.

TABELA 5.8 HORAS GASTAS COM TESTES X HORAS GASTAS COM MANUTENÇÃO

Teste	Manut < 4 h	Manut 4 a 8 h	Manut 8 a 16 h	Manut > 16 h	Total
Branco	1	0	0	0	1
Total %	2,08%	0,00%	0,00%	0,00%	2,08%
< 4 h	6	2	1	1	10
Total %	12,50%	4,17%	2,08%	2,08%	20,83%
4 a 8 h	7	6	3	0	16
Total %	14,58%	12,50%	6,25%	0,00%	33,33%
8 a 16 h	6	4	2	2	14
Total %	12,50%	8,33%	4,17%	4,17%	29,17%
> 16 h	3	1	2	1	7
Total %	6,25%	2,08%	4,17%	2,08%	14,58%
Total	23	13	8	4	48
Total %	47,92%	27,08%	16,67%	8,33%	

Outro fato verificado e observado na Figura 5.20 é que 60,4% das pessoas acham que necessariamente devem ser realizados testes segundo uma metodologia e com muito critério, logo têm consciência da importância do teste.

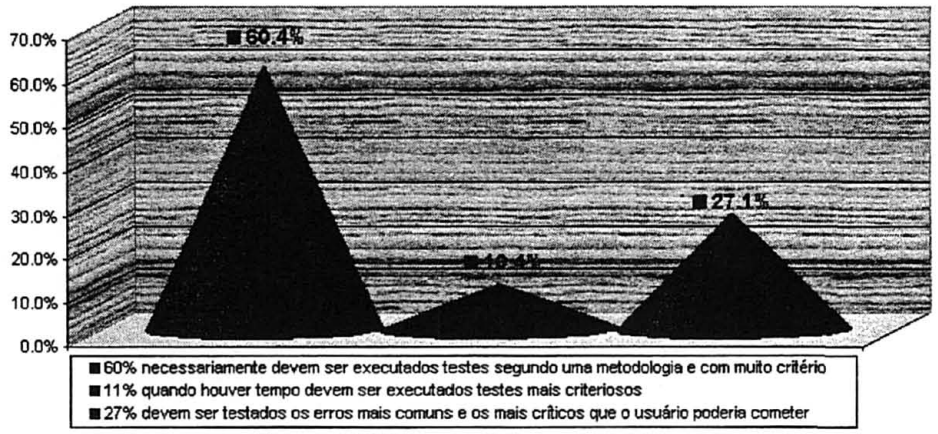


FIGURA 5.20 COMO DEVE SER FEITO O TESTE

Se são gastas muitas horas com testes como foi demonstrado anteriormente e a maioria das pessoas têm consciência da importância da realização dos testes com critério, alguma coisa está errada, pois foi identificado na Figura 5.13 que mais da metade das pessoas (56,3%) encontram erros de código nos programas e 39,6% encontram problemas de desempenho. Uma das causas pode ser, se analisada a Figura 5.21, que Às vezes ou Nunca a realização desta atividade é conduzida de uma forma sistemática e organizada.

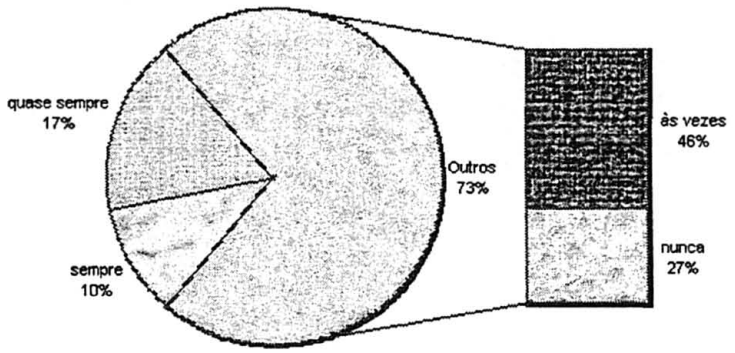


FIGURA 5.21 CONDUÇÃO DOS TESTES COMO UMA ATIVIDADE SISTEMÁTICA E ORGANIZADA

E também pode ser observado pela Figura 5.22, que 45,8% Às vezes e 6,3% Nunca costumam fazer um plano de teste estabelecendo objetivos e respondendo às perguntas "o que testar" e "quando terminar".

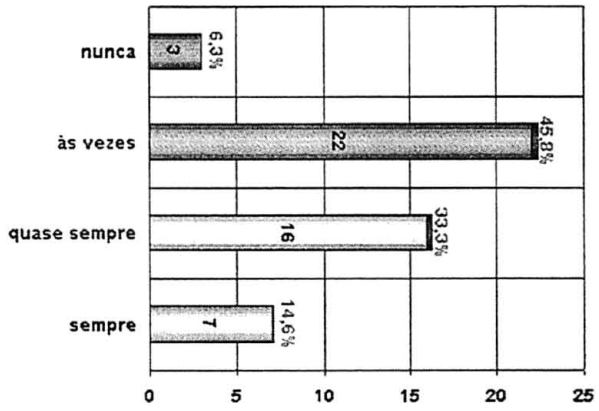


FIGURA 5.22 DESENVOLVIMENTO DE UM PLANO DE TESTE

5.2 CONSIDERAÇÕES FINAIS

Considerando a análise dos resultados, pode-se concluir que a estratégia de teste a ser proposta, deverá considerar ou incluir os seguintes aspectos:

1) Fazer com que os testadores estejam motivados a encontrar erros nos programas. Esse deve ser o objetivo principal de todo o teste, consequentemente garante que o produto resultante atenda às necessidades funcionais e comportamentais (desempenho, segurança, etc.) esperados.

2) Proporcionar meios de testar o desempenho do sistema, pois foi verificado que esse teste quase não é feito e procurar proporcionar meios de simular o ambiente final com carga de dados e número de usuários.

3) Indicar uma maneira eficaz de registrar os recursos de tempo e os defeitos encontrados durante uma atividade de teste para se obter um relação custo-benefício desta atividade e justificar a importância da contemplação de um tempo específico para ela no cronograma do projeto.

4) Contemplar uma maneira de manter arquivado os dados de teste usados, podendo atualizá-los a fim de manter um histórico e poderem ser usados no teste de regressão, ou seja, um ambiente de teste.

5) Outras necessidades apontadas: realizar mais testes estruturais, prover um meio de auxiliar na aplicação dos testes funcionais com dados gerados a partir dos requisitos do usuário e propor meios mais eficientes de geração de dados.

6) Comprovadas a necessidade e expectativa de um roteiro, o que pode ser simplista demais, que auxilie a atividade de teste, e a maior expectativa é que a utilização de uma ferramenta auxilie a execução desta tarefa. No caso da ferramenta este trabalho procura relacionar ferramentas disponíveis comercialmente que podem auxiliar esta tarefa embora a ETACS possa ser executada sem o auxílio de uma.

7) Como a maioria concordou embora não pratiquem, devem ser executados testes segundo uma metodologia e com muito critério como uma atividade sistemática e organizada fazendo um plano de teste. Falta conhecimento formal de métodos de teste, atualmente é intuitivo.

Com isso, conclui-se que é imprescindível a elaboração de uma estratégia de teste para auxiliar esta atividade na empresa.

6 A ESTRATÉGIA PROPOSTA – ETACS

A ETACS (**E**stratégia de **T**este de Software para **A**mbiente **C**liente-**S**ervidor) é uma estratégia voltada para ambiente cliente-servidor adaptada de estratégias, processos, metodologias conhecidos na literatura apresentados no Capítulo 4. Ela foi proposta considerando o levantamento das necessidades reais obtidos a partir de uma pesquisa realizada em uma empresa de médio porte de desenvolvimento de sistemas descrita no capítulo anterior.

Para efeitos de elaboração desta estratégia foram consideradas as seguintes características de ambiente e padrões de desenvolvimento:

1. Ambiente cliente-servidor 3 camadas, conforme descrito no Capítulo 2.
2. Ciclo de vida de desenvolvimento de software iterativo (Figura 4.4) [29],[59], descrito na Sub-seção 4.5.
3. Caso de uso para capturar e definir os requisitos funcionais do sistema de software, [29],[59].
4. A ETACS pode ser aplicada sem o uso de uma ferramenta. Entretanto é altamente recomendada a adoção de um suporte automatizado por razões já apontadas na Seção 3.4. Para tanto, o APÊNDICE A, apresenta uma relação de grandes empresas da área de informática que fornecem ferramentas de teste que podem ser aplicadas em ambiente cliente-servidor e que podem ser utilizadas conjuntamente com a ETACS.

Uma visão geral e estática da ETACS pode ser vista na Figura 6.1, que apresenta 6 grandes etapas, que podem ocorrer iterativamente conforme a necessidade do sistema.

A Figura 6.1 apresenta uma visão estática do ciclo das etapas do teste durante o desenvolvimento. Em cada etapa é realizada uma série de atividades. Para auxiliar na atividade de avaliação dos riscos e prioridade e na atividade de identificação dos tipos de testes, a ETACS propõe dois roteiros descritos nas Seções 6.7 e 6.8 respectivamente. A ETACS prevê a realização de cada etapa como descrito nas seções subseqüentes.

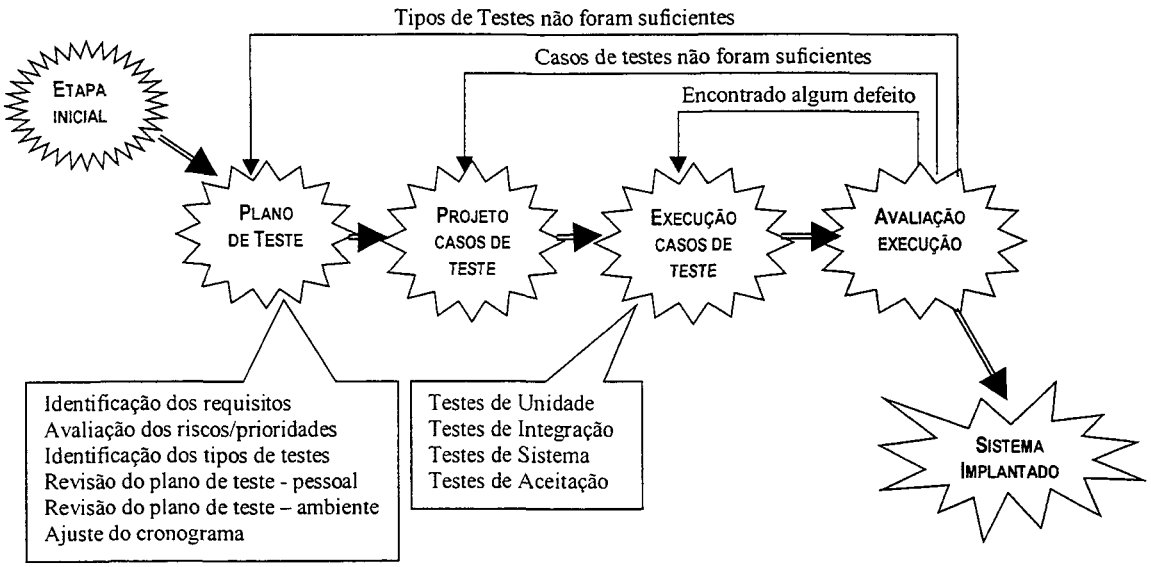


FIGURA 6.1 VISÃO GERAL DO CICLO DO TESTE DA ETACS

6.1 ETAPA INICIAL

Esta etapa é também conhecida como modelagem de negócio, onde para efeitos da atividade de teste são definidas apenas as estimativas de custo para esta atividade. Para que possa ser gerada esta estimativa de custo, outras informações do sistema são necessárias além do principal que é a definição do problema, qualquer outra informação pode servir de auxílio para este levantamento.

Nesta etapa são geradas então várias informações iniciais, que dão entendimento sobre o projeto de sistema que deverá ser desenvolvido. A estimativa de custo de teste deve fazer parte dessas informações além de:

- conjunto de atividades e tarefas a serem executadas para o projeto;
- visão global da arquitetura a ser utilizada para o sistema;
- requisitos levantados em casos de usos sem muito detalhe mas que possibilite uma visão geral do sistema e do qual possam ser verificados que tipos de testes devem ser realizados;
- identificação da demanda a ser atendida, destacando os principais elementos envolvidos, tais como: abrangência, necessidades, agentes envolvidos, requisitos e outras variáveis relevantes que proporcionem uma visão geral da demanda;
- descrição da solução sucinta e objetiva que deverá sempre possibilitar uma análise de benefícios e custos, avaliando-se assim, qual das alternativas é a mais adequada,

e de que forma essa solução contribuirá para a resolução da situação demandada e proporcionará ganhos para o cliente;

- se o sistema vai interagir com outros já existentes;
- uma estimativa de qual ou quais recursos humanos, ambiente de teste e *softwares* que serão requeridos, para que a atividade de teste do sistema seja executada e estimativa de custos associados;
- deverá ser formulado um Plano de Trabalho que conterá as tarefas a serem desenvolvidas com os papéis e responsabilidades definidos para os técnicos da equipe, além do tempo, custo e qualidade a serem alcançados quando da aplicação deste plano e o registro dos itens mencionados anteriormente;
- aprovação do projeto a ser realizado com o plano de trabalho junto ao cliente.

Essas informações são utilizadas como entrada de dados para gerar o plano de teste, conforme descrito a seguir.

6.2 PLANO DE TESTE

Depois que as informações iniciais já foram levantadas, conforme descrito na etapa inicial, pode-se iniciar a construção do plano de teste.

O plano de teste deve incluir: identificação de requisitos, avaliação dos riscos e definição das prioridades, identificação dos tipos de testes, técnicas e critérios que serão utilizados, revisão do plano de trabalho quanto aos recursos humanos e ambiente.

6.2.1 Identificação dos requisitos

Todas as informações que o usuário transmita devem ser registradas. A partir dessas informações devem ser identificados os requisitos funcionais e não funcionais.

Requisitos funcionais

Os casos de usos representam a funcionalidade do software, "o que" ele produz. A partir desta identificação serão definidos quais são os testes que serão aplicados para cada caso de uso. Um caso de uso pode gerar nenhum, um ou mais dados de teste. Para complementar os casos de teste, além dos casos de uso, são utilizadas também entrevistas com o usuário-final e outras documentações levantadas para o projeto.

Requisitos não funcionais

Devem também ser identificados, de forma narrativa, os requisitos não funcionais, que não podem ser mapeados através dos casos de uso, principalmente os relacionados com segurança, desempenho e precisão que no ambiente cliente-servidor têm maior impacto.

6.2.2 Avaliação dos riscos e definição de prioridades

Depois que os casos de uso estão especificados é necessário identificar quais têm maior prioridade. A prioridade é diretamente proporcional ao relacionamento entre os casos de uso, e tem a finalidade de classificar os casos de uso para aplicação de critérios mais rigorosos e rígidos e de maior custo quando forem necessários. Para auxiliar nesta tarefa, que é bastante complexa e importante, é proposto um roteiro apresentado na Seção 6.7.

6.2.3 Identificação dos tipos de testes, técnicas e critérios

Com base nas prioridades, devem ser aplicados diferentes testes, considerando que quanto mais testar mais caro fica e que testes aleatórios não podem ser medidos, portanto, os critérios de testes aplicados são mais eficazes do que testes aleatórios (geração de dados sem critério), esta estratégia apresenta alguns exemplos de testes e propõe uma sugestão de técnicas e critérios de acordo com o grau de prioridade do caso de uso. Nesta tarefa é importante identificar testes para as diferentes camadas do ambiente cliente-servidor. Para auxiliar nesta tarefa a ETACS propõe um roteiro apresentado na Seção 6.8.

6.2.4 Revisão do plano de trabalho quanto aos recursos humanos

Neste momento é possível estimar um custo mais aproximado do real, pois já se sabe quais testes serão aplicados e quais os recursos humanos que serão necessários para que estes sejam realizados. Definem-se as pessoas que terão as responsabilidades pelas atividades de teste, o papel de cada uma, quem vai gerenciar, quem vai gerar os casos de teste, quem vai executar, quem vai providenciar o ambiente, quem vai documentar.

6.2.5 Revisão do plano de trabalho quanto ao ambiente

Nesta etapa deve ser feita uma revisão no plano de trabalho quanto à estimativa de custos para o ambiente em termos de software e *hardware*, pois como já foram identificados quais os critérios, técnicas, pessoas que irão realizar todas as etapas do teste, pode-se reavaliar os custos dentro da realidade. Devem ser levantadas informações como: qual banco de dados será utilizado para os testes, providências a serem tomadas para que sejam

disponibilizadas chaves, senhas e diretórios para armazenamento dos arquivos, máquinas, ferramentas de teste, etc.

6.2.6 Ajuste do cronograma

Para o bom desenvolvimento de um sistema, deve-se ter um cronograma para que o desenvolvimento do projeto possa ser melhor gerenciado, estabelecendo marcos e produtos intermediários, minimizando a chance de fracasso. Depois de levantadas as informações no Plano de Teste com relação aos tipos de testes, pessoas e ambientes, é possível ajustar o cronograma do projeto contemplando as atividades destas pessoas bem como os produtos intermediários com relação aos testes.

6.3 PROJETO DE CASOS DE TESTE

A partir dos casos de uso devem-se extrair casos de teste e depois, dependendo do critério e técnica, selecionar os casos de testes mais adequados. Cada critério e tipo de teste determina como são gerados os casos de teste. A funcionalidade do software está contemplada pois a geração dos dados de teste é feita através dos casos de uso e os requisitos não funcionais, que são apenas apontados devem afetar principalmente a etapa de execução e avaliação dos casos de teste.

Os casos de teste podem ser projetados desde a fase de análise para testes mais funcionais, e depois adequados conforme a necessidade. O importante é que em algum momento destas iterações, depois de definidos os tipos de testes que serão realizados, sejam especificados os critérios de satisfação para cada tipo de teste, ou seja critério de cobertura. Os critérios de geração de dados de teste normalmente já indicam critérios de cobertura (Capítulo 3).

A Tabela 6.1, baseada na tabela apresentada por Mosley [45], apresenta uma maneira de mesclar os níveis de teste [52] dentro do ciclo tradicional de desenvolvimento, mostrando em que níveis os casos de teste são elaborados e executados. Como pode ser observado, os casos de teste para os testes de sistema e aceitação já podem ser projetados na fase de análise, assim que os requisitos (funcionais e não funcionais) são levantados e, serão executados praticamente em uma fase em que o sistema esteja pronto ou parcialmente pronto, com as partes integradas, para que possa ser testado o requisito levantado.

TABELA 6.1 NÍVEIS DE TESTE DENTRO DO CICLO DE DESENVOLVIMENTO TRADICIONAL

Fases do ciclo de vida de desenvolvimento	Plano de teste produzido	Plano de teste executado
Análise	Sistema/Aceitação	
Análise na revisão	Regressão	
Projeto de sistema	Integração	
Construção	Unidade	Unidade/Integração
Teste		Sistema/Aceitação
Produção/Manutenção		Regressão

Adequando estes níveis de teste ao modelo iterativo utilizado pela ETACS e conforme descrições apontadas na Seção 4.5, pode-se definir a Tabela 6.2 considerando que, na fase de inicialização onde é definido o escopo e viabilidade da solução do problema e onde são especificados os requisitos iniciais, já devem ser planejados os casos de teste para aceitação, com base nestes requisitos. Na fase de elaboração, onde estes requisitos são representados em casos de uso e são mais detalhados, devem ser produzidos casos de teste de sistema e como o ciclo é iterativo, se houver mais de uma iteração são reutilizados os dados de teste (regressão). Na fase de construção, o que corresponde a fase de projeto e construção no ciclo tradicional, devem ser projetados os casos de teste para testes de integração e unidade e executando-os. Também nesta fase devem ser executados os testes de sistema projetados e quando este módulo estiver pronto, verificar se ele atende a todos os requisitos (aceitação) durante a fase de transição.

TABELA 6.2 NÍVEIS DE TESTE DENTRO DO CICLO DE DESENVOLVIMENTO ITERATIVO

Fases do ciclo de vida de desenvolvimento	Plano de teste produzido	Plano de teste executado
Inicialização	Aceitação	
Elaboração	Sistema/Regressão	
Construção	Integração /Unidade	Unidade/ Integração/Sistema
Transição		Sistema/Aceitação

6.4 EXECUÇÃO DOS CASOS DE TESTE

As Tabela 6.1 e Tabela 6.2 apresentam em que etapas do desenvolvimento a execução dos casos testes será realizada. A ETACS é aplicada utilizando o ciclo de vida iterativo, onde as atividades inerentes aos testes são realizadas durante todas as fases. É na fase de construção que se concentra o maior esforço, mais tempo do teste, conforme observado na Figura 4.4.

Outro detalhe é que os testes de unidade e integração são projetados e executados na fase de construção, pois é necessário ter código e interface bem definidos para estes tipos de testes. Para a execução dos testes de unidade e integração pode ser utilizado o depurador que as linguagens de desenvolvimento disponibilizam para verificação e garantia da

execução correta das instruções dentro do algoritmo, bem como a passagem correta de parâmetros.

Considerando uma arquitetura cliente-servidor três camadas, alguns testes devem ser considerados essenciais: os testes de sistema em geral, mas principalmente desempenho, estresse e carga. A ETACS faz uma série de considerações para esta etapa, a qual é considerada a mais importante, por se ela a mais influenciada pela arquitetura cliente-servidor. Devem ser implementados 2 a 5 casos de usos selecionados arbitrariamente, incluindo consulta e atualização do banco (quando houver no sistema) para validar a arquitetura em que será construído o sistema. Para sistemas mais complexos podem ser escolhidos até mais casos de uso de acordo com a percepção do analista de negócio e gerente do projeto.

O desempenho é característica de um requisito não funcional, que é parte tangível da qualidade final do produto. Um software pode fazer exatamente como foi planejado, todavia, lento demais a ponto de inviabilizá-lo. Por isso deve ser validado nas primeiras iterações através do teste de desempenho, que normalmente é realizado juntamente com o teste de estresse e carga. A mudança de arquitetura é cara se alterada tarde e é importante não somente descobrir se ela resolve o problema mas também se resolve o problema com um bom desempenho. A importância que a arquitetura tem para o desempenho foi apresentada em detalhes no trabalho de Martins [39] e no livro do Pattison [49].

Na arquitetura cliente-servidor os tempos para a execução dos componentes podem variar muito devido a várias causas. Como foi visto no Capítulo 5, desempenho foi um dos itens de maior problema apontado, que pode ser solucionado pela execução correta dos testes de sistema. Uma das razões apontadas para que estes testes não sejam feitos foi a dificuldade de simular o ambiente do cliente. Mas o que ocorre na realidade é que os testes são realizados superficialmente, com apenas um ou no máximo dois usuários acessando, utilizando uma ou duas máquinas e quando o sistema é colocado em produção, a realidade é outra. São vários usuários acessando e dependendo da arquitetura utilizada o desempenho pode cair muito com o aumento da escalabilidade, principalmente com a arquitetura 2 camadas.

Na execução do teste de desempenho juntamente com estresse, deve-se utilizar 6 máquinas rodando simultaneamente. Este número foi obtido por verificar que quando tem-se mais de 5 computadores consegue-se obter diferença considerável nos tempos obtidos para execução nas diferentes arquiteturas. Com relação ao tempo deve-se fazer uma comparação entre o tempo de execução do componente e apresentação dos dados na

interface a fim de isolar a camada de apresentação da camada de negócio. E isolar a camada de dados acessando diretamente o banco de dados.

Outro detalhe importante neste tipo de arquitetura, utilizando componentes com um monitor de transação (típico da arquitetura 3 camadas), é que, para simular o ambiente do cliente, ou melhor, simular uma execução mais próxima da realidade, devem-se levar em conta as características do software a ser produzido. Se o software será utilizado de uma forma similar ao estresse, como atualizações com sistema *batch*, central de atendimento, onde a utilização é grande ou o sistema é até utilizado por várias pessoas mas elas acessam o sistema depois de um tempo acessam novamente. Isso faz uma grande diferença neste tipo de arquitetura.

Se a característica do software é voltada para uma característica de estresse, deve-se necessariamente simular esta situação. Deve-se executar os componentes em intervalos de tempo, a cada dez minutos por exemplo. Há toda uma lógica para disponibilizar o componente, controlado pelo monitor de transação, que gerencia os recursos conforme a necessidade. Se algum componente fica ocioso, o monitor o tira da memória para otimizar o desempenho do servidor. Dez minutos são suficientes para que o monitor de transação reutilize os recursos e quando os componentes são acessados novamente o tempo para executá-los é bem diferente de quando executado sucessivamente.

O monitor de transação, como o próprio nome já diz gerencia as transações. E gerencia também as conexões com o banco de dados, otimizando-as, para isto é necessário que a arquitetura seja configurada para tal.

6.5 AVALIAÇÃO DOS CASOS DE TESTES IMPLEMENTADOS

De acordo com as métricas apresentadas no Capítulo 3 de revisão sobre teste pode-se verificar se os casos de testes aplicados foram suficientes para atingir a cobertura necessária para cada critério. Se não foram suficientes, novos casos de teste devem ser gerados. Conforme observado na Figura 5.1, se encontrado algum defeito, após a sua correção, testes de regressão devem ser realizados.

Para avaliar possíveis problemas de desempenho na arquitetura cliente-servidor, devem-se analisar os comparativos entre o tempo que o componente leva para executar e o tempo que a interface leva para apresentar juntamente. E também comparar com o tempo que levaria se fosse acessado diretamente o banco de dados. Se este último tempo comparado com os outros for alto, provavelmente pode haver uma melhoria na camada de dados,

O estabelecimento das prioridades, para gastar mais esforço de teste com o que tem mais prioridade, está fortemente baseado no impacto que um defeito ou falha pode provocar no software analisado sob vários aspectos conforme apresentado na na Seção 4.5.1 proposto pela ferramenta da Rational [57],[43].

No entanto, algumas modificações foram feitas, pois é difícil analisar os casos de uso segundo os conceitos da Tabela 4.4 [32], [57], uma vez que são bastante genéricos, por isso e embasado nos estudos de priorização de requisitos discutidos por Karksson, Ryan e Olsson em [30],[74] onde apresentam uma tabela (Tabela 6.3), para o assinalamento numérico conforme o grau de importância dos requisitos seguindo uma escala, foi adotada a Tabela 4.3, utilizada pela metodologia do Test-Rx [45], que mais se enquadra nos conceitos propostos pela Tabela 6.3. Ou seja será utilizada a metodologia proposta pela Rational mas a tabela de referência de conceitos será utilizada do processo Test-Rx.

TABELA 6.3 ESCALA PARA ATRIBUIÇÃO DE IMPORTÂNCIA ABSOLUTA [30]

Importância	Definição
1	não faz falta
2	não importante (o cliente poderá aceitar sua ausência)
3	Preferencialmente importante (o cliente poderá apreciá-lo)
4	Muito importante (o cliente não deseja ficar sem ele)
5	Mandatória (o cliente não pode ficar sem ele)

Os conceitos da Tabela 4.3 devem ser aplicados a cada caso de uso, de acordo com as situações que estão sendo avaliadas. Estas situações estão resumidas na Tabela 6.4 e são baseadas no estabelecimento de requisitos descrito na Seção 4.5.1 e detalhadas na seqüência. Os casos de uso devem ser listados, avaliados e justificados de acordo com estas situações.

TABELA 6.4 DESCRIÇÃO RESUMIDA DOS ITENS A SEREM AVALIADOS NOS RISCOS E SEUS PESOS

Item	Descrição abreviada	Pesos
1	Efeitos de uma falha no caso de uso	3
2	Causas de uma falha no caso de uso	3
3	Probabilidade do caso de uso falhar	3
4	Número de acessos a este caso de uso	2
5	Perfil dos usuários que utilizarão o caso de uso	1
6	Contrato com fornecedor deste caso de uso	3

Será analisado cada um dos itens da Tabela 6.4:

1) **Efeitos de uma falha no caso de uso** - o que aconteceria se algo desse errado neste caso de uso? Quais os impactos ou conseqüências de uma falha neste caso de uso? Quais os efeitos da falha? Por exemplo: O que aconteceria se a aquisição de um produto via *internet* tivesse algum problema? Deve ser utilizada a Tabela 4.3 como balizadora para atribuir um grau e justificar porque foi atribuído este grau, porque foi escolhido um grau

crítico, regular ou outro. A Tabela 6.5 apresenta um exemplo de justificativa para causa e da mesma forma deve ser justificado com outro enfoque: o efeito.

2) **Causas de uma falha no caso de uso** - Como poderia algo dar errado, o que provocaria uma falha? Quais as possíveis causas de falhas? Por exemplo, Como poderia uma transação não ser refletida na base de dados central? Quais as possíveis causas para que uma aquisição de um produto via *internet* não fosse concretizada? Deve ser utilizada a Tabela 4.3 como balizadora e justificado conforme exemplo da Tabela 6.5.

TABELA 6.5 EXEMPLO DE JUSTIFICATIVA DE CAUSA DE UMA FALHA NO CASO DE USO

Caso de uso	Causa	Grau do efeito	Justificativa
Aquisição do produto via *internet*	Conexão da *internet* perdida enquanto incluindo itens de compra	Regular	Necessário fazer algumas programações no sistema para não ficar com dados inconsistentes caso isto aconteça, mas se não for possível fazer a compra por *internet* pode ser feita por telefone ou pessoalmente.

3) **Probabilidade do caso de uso falhar** - Qual a probabilidade deste caso de uso falhar? De ocorrer uma tempestade e cair a energia? Do disco rígido ser danificado? Dos valores entrados estarem fora do limite? Das causas que provocariam uma falha neste caso de uso? Quanto maior a probabilidade do caso de uso falhar, maior deve ser o conceito atribuído.

4) **Número de acessos a este caso de uso** - Este caso de uso é utilizado por quantos usuários? Qual o número de vezes que este caso de uso é executado em um dado período de tempo (no mês)? Para facilitar, pode ser imaginado um número máximo possível de utilização por dia, e de usuários acessando simultaneamente. Divide-se este valor por 5 e estabelece-se o maior para o tipo "Crítico", e ou outros para demais faixas ajustando-se conforme a Tabela 4.3 e as orientações a seguir:

- **Crítico** para casos de uso que acontecem com muita freqüência no período ou utilizados por muitos atores simultaneamente;
- **Prioritário** para casos de uso que ocorrem com muita freqüência, muitas vezes no período que envolvem sempre os mesmos usuários – pois minimiza a possibilidade de colisão;
- **Regular** para casos de uso usados em uma escala menor, com certa freqüência, várias vezes no período;
- **Pouca importância** baixa ocorrência e que envolvem poucos atores; e
- **Exceção de qualidade** para casos de usos que raramente ocorrem e envolvem poucos atores.

5) **Perfil dos usuários que utilizarão o caso de uso** - Como sugerido pela ferramenta RUP da Rational (Sub-seção 4.5.1) [29], que estabelece que deve ser levada em consideração a experiência e tolerância para o perfil do usuário que vai usar o sistema ETACS propõe analisar o perfil do usuário que vai utilizar o sistema principalmente quanto a sua experiência, pois isto pode impactar na quantidade de defeitos que ele pode provocar. E em segundo plano, a tolerância.

Os usuários experientes tendem a colocar os valores corretos e não fazer coisas não esperadas. Mas também deve ser levada em consideração a exigência do usuário, pois os usuários podem ser classificados em dois tipos, um mais complacente à falha, não gerando por isso, grande impacto na imagem da empresa e, outro, muito exigente para o qual uma falha pode afetar muito a imagem da empresa.

O perfil do usuário pode ser classificado segundo as orientações a seguir:

- **Crítico** - usuário inexperiente e não complacente quanto as falhas
- **Prioritário** - usuário inexperiente mas que tolera algum tipo de falha
- **Regular -** o usuário pouco experiente e não tolerante a falha
- **Pouca importância** - usuário experiente exigente e que não tolera falha
- **Exceção de qualidade -** usuário experiente tolerante a falha

6) **Contrato com fornecedor deste caso de uso** – É proposto que para analisar o contrato com o fornecedor, seja verificado se o sistema pode ser entregue sem este caso de uso em questão, conforme as orientações a seguir:

- **Crítico** - se não tiver como entregar alguma versão sem este caso de uso; ele é parte essencial do sistema;
- **Prioritário** – se o sistema se torna sem importância e sem sentido sem este caso de uso mas é possível entregar uma versão sem ele;
- **Regular** - se for possível entregar uma versão parcial sem este caso de uso, mas isso faz uma diferença aceitável na versão;
- **Pouca importância** - é possível entregar o sistema sem este caso de uso e isso não faz muita diferença; e
- **Exceção de qualidade** - não faz diferença alguma o fato deste caso de uso não ser entregue, em muitos casos existem formas manuais práticas de resolver este caso de uso que não causaria grande impacto para o sistema.

A Tabela 6.3 apresenta os cinco conceitos estabelecendo um valor para cada um deles. Os conceitos da Tabela 4.3, adotados pela ETACS para avaliação dos 6 itens (Tabela 6.4), são similares e foram atribuídos os mesmos valores. Dessa associação foi gerada a Tabela 6.6, que serve de base para a classificação dos casos de uso.

TABELA 6.6 VALORES PARA ATRIBUIÇÃO DA PRIORIDADE

Definição	Valor
Crítico (Cr)	5
Prioritário (Pr)	4
Regular (Re)	3
Pouca importância (Po)	2
Exceção de qualidade (Ex)	1

6.7.1 Aplicação do roteiro em um exemplo fictício

Procurando lapidar qualquer aresta que possa ter ficado com a apresentação do roteiro, é apresentado um exemplo mostrando como são aplicados estes conceitos.

Supondo 4 casos de usos, analisando-se cada item da Tabela 6.4 para cada caso e atribuindo-se os conceitos conforme descrito até aqui é obtida a Tabela 6.7. Observando-se que para o caso de uso 1 o Item 3 não se aplica.

TABELA 6.7 EXEMPLO DE TABELA COM OS CONCEITOS PARA OS ITENS (ENTRADA DA PLANILHA)

Descrição do caso de uso	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Caso de Uso 1	Crítico	Prioritário		Pouca importância	Regular	Exceção de qualidade
Caso de Uso 2	Regular	Pouca importância	Prioritário	Crítico	Regular	Regular
Caso de Uso 3	Prioritário	Regular	Pouca importância	Regular	Exceção de qualidade	Regular
Caso de Uso 4	Regular	Pouca importância	Exceção de qualidade	Pouca importância	Crítico	Exceção de qualidade

De acordo com o valor da Tabela 6.6, os conceitos são substituídos por valores correspondentes, conforme a Tabela 6.8 apresenta.

TABELA 6.8 TABELA INTERMEDIÁRIA DE EXEMPLO DOS CONCEITOS X ITENS

Descrição do caso de uso	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
Caso de Uso 1	Cr=5	Pr=4	nulo	Po =2	Re =3	Ex =1
Caso de Uso 2	Re=3	Po=2	Pr =4	Cr =5	Re =3	Re =3
Caso de Uso 3	Pr =4	Re =3	Po =2	Re =3	Ex =1	Re =3
Caso de Uso 4	Re =3	Po =2	Ex=1	Po =2	Cr =5	Ex =1

Os itens 1, 2 e 3 estão extremamente relacionados e não necessariamente coexistem. A própria ferramenta da Rational [32],[57] propõe a união destes 3 itens utilizando o maior conceito. A ETACS propõe que seja feita a média dos itens que forem identificados, obtendo-se os resultados mostrados na Tabela 6.9. Por exemplo, para o Caso de Uso 1, a

média será 4,5 (item 1 = 5 + item 2 = 4 + item 3 = nulo=desconsiderado) 5 + 4=9 dividido por 2 (pois só existem dois itens não nulos), resulta em 4,5.

TABELA 6.9 TABELA INTERMEDIÁRIA COM MÉDIA DOS ITENS 1, 2 E 3

	Item 1,2,3	Item 4	Item 5	Item 6
Descrição do caso de uso / peso ->	**3**	**2**	**1**	**3**
Caso de Uso 1	4,50	2	3	1
Caso de Uso 2	3,00	5	3	3
Caso de Uso 3	3,00	3	1	3
Caso de Uso 4	2,00	2	5	1

Continuando com o exemplo, o valor 4,5 multiplicado pelo peso dos itens (1,2 e 3) que é 3 conforme a Tabela 6.4, resulta em 13,5, apresentado na Tabela 6.10.

Para os demais itens, item 4, item 5 e item 6, a aplicação é direta, não é necessário fazer a média antes. Como no Caso de Uso 1, item 4, multiplica-se o valor 2 pelo peso correspondente ao item 4 que é 2, (Tabela 6.9) e produz-se o resultado 4 (Tabela 6.10).

TABELA 6.10 TABELA INTERMEDIÁRIA CONSIDERANDO O PESO DOS ITENS

	Item 1,2,3	Item 4	Item 5	Item 6	Somatório
Descrição do caso de uso	**peso=3**	**peso=2**	**peso=1**	**Peso=3**	
Caso de Uso 1	13,50	4,00	3,00	3,00	23,50
Caso de Uso 2	9,00	10,00	3,00	9,00	31,00
Caso de Uso 3	9,00	6,00	1,00	9,00	25,00
Caso de Uso 4	6,00	4,00	5,00	3,00	18,00

Depois de fazer a multiplicação dos valores pelos pesos para todos os itens, somam-se os valores calculados com seus pesos e produz uma coluna *Somatório* conforme mostra a Tabela 6.10. Tem-se a seguinte fórmula para se calcular o coeficiente final (Cf):

*Cf=∑((média absoluta dos item_1, item_2, item_3) * 3 + item_4 * 2 + item_5 * 1 + item 6 * 3)*

Utiliza-se o valor deste coeficiente para se definir a prioridade do caso de uso tal como apresentada na Figura 6.2. Estes coeficientes também são utilizados pela ferramenta da Rational [57].

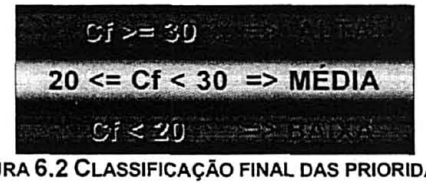


FIGURA 6.2 CLASSIFICAÇÃO FINAL DAS PRIORIDADES

Se dois casos de uso têm o mesmo conceito de prioridade, verifica-se qual tem maior coeficiente (Cf). Dentre os casos de mesmo conceito de classificação final e mesmo coeficiente, pode-se estabelecer um dos itens como sendo o classificador entre eles.

A Tabela 6.10 foi classificada em ordem decrescente do somatório e atribuído o conceito final segundo a Figura 6.2, produzindo a Tabela 6.11. Segundo esta tabela, devem ser

aplicados critérios mais rigorosos quanto maior for o conceito final para o caso de uso. Para auxiliar na seleção dos critérios foi proposto o roteiro apresentado na próxima seção.

TABELA 6.11 TABELA FINAL DE EXEMPLO DAS PRIORIDADES

Descrição do Caso de uso	Item (1,2,3) (x 3)	Item 4 (x 2)	Item 5 (x 1)	Item 6 (x 3)	Somatório dos Valores obtidos	Prioridade
Caso de Uso 2	9,00	10,00	3,00	9,00	31,00	
Caso de Uso 3	9,00	6,00	1,00	9,00	25,00	Média(1)
Caso de Uso 1	13,50	4,00	3,00	3,00	23,50	Média(2)
Caso de Uso 4	6,00	4,00	5,00	3,00	18,00	Baixa

6.8 ROTEIRO PARA IDENTIFICAÇÃO DOS TIPOS DE TESTES, TÉCNICAS E CRITÉRIOS

Esta identificação pode ser refinada ao longo da atividade, numa primeira etapa devem ser informados os principais testes e normalmente os que geralmente são realizados.

Portanto, como cada critério de teste (ver Capítulo 3) pode identificar grupos de defeitos diferentes, é importante uma combinação destes critérios para que a qualidade da atividade de teste seja atingida. Se for aplicado apenas um critério, a tendência é que um grupo de defeitos seja encontrado e talvez outros não.

Critérios de teste mais rigorosos são, em geral, mais caros e, portanto, devem ser aplicados em casos de uso que justifiquem este custo. Nesse sentido propõe-se a utilização da hierarquia dada pela relação entre critérios (Seção 3.3).

Na primeira etapa, a de inicialização, refere-se a etapa inicial da ETACS, só são definidos os testes mais utilizados conforme as características principais do sistema. Na etapa de elaboração inicia-se o plano de teste e devem ser relacionados os tipos de testes que serão utilizados, para que sejam gerados casos de teste específicos para cada tipo de teste. Mas no início só serão identificados quais os tipos de teste e quais casos de uso devem utilizá-lo, posteriormente com detalhes sobre os casos de uso, pode-se eliminar ou incluir mais testes ou casos de teste para aqueles já identificados (Figura 6.1).

6.8.1 Níveis de teste

A atividade de teste é realizada em níveis diferentes durante o ciclo de vida do software. O objetivo é encontrar erros; diferentes tipos de erros são procurados em cada nível [15]. A estratégia genérica de teste proposta por Pressman, (Seção 4.1), propõe 4 níveis: Teste de Unidade, Teste de Integração de Componentes, Teste de Sistema e Teste de Aceitação.

A ETACS combinou os tipos de testes para cada camada do ambiente cliente-servidor proposta por Mosley com os níveis propostos por Pressman e foi elaborada a Tabela 6.12, que apresenta uma sugestão de quais níveis devem ser testados em cada camada do ambiente cliente-servidor.

TABELA 6.12 DISTRIBUIÇÃO DOS NÍVEIS DE TESTE DENTRO DE CADA CAMADA

Níveis de teste	Camada de apresentação	Camada de negócio	Camada de dados
1. Teste de unidade		X	
2. Teste de integração	X	X	X
3. Teste de sistema	X	X	X
4. Teste de aceitação	X		
5. Teste de regressão	X	X	X

6.8.1.1 Camada de apresentação

Na camada de apresentação, onde são colocadas as lógicas de apresentação, podem ser feitos alguns testes de unidade, pois podem existir lógicas mais complicadas, mas não é comum, por isso normalmente não é necessário teste de unidade para esta camada. O teste de integração é necessário, pois um sistema em camadas pressupõe que o cliente aciona um componente e é preciso verificar se as interfaces estão bem definidas, se o que o cliente está enviando é realmente o que o componente está recebendo e vice-versa.

O teste de aceitação deve ser bem criterioso na camada de apresentação, pois são analisadas questões de interface (usabilidade, se os ícones estão bem representativos, se as cores contrastam corretamente) e requisitos funcionais e não funcionais de acordo com a especificação. Portanto podem ser analisadas como protótipo, só interface e com todas as camadas, funcionalidade. Os dados dos testes anteriores podem ser reaproveitados e por isso o teste de regressão é também executado nesta camada.

O teste de sistema é bem importante quando executado com todas as camadas para verificação de desempenho, e é necessário e importante, isolar o tempo que o componente leva para executar a transação e o tempo total incluindo o tempo que a camada de apresentação leva para mostrar as informações obtidas. Mais detalhes sobre este tipo de teste estão descritos na Seção 6.4.

6.8.1.2 Camada de negócio

Na camada de negócio, onde são colocadas as lógicas do negócio, numa arquitetura 3 camadas utilizando um monitor de transação, podem ser feitos alguns testes de unidade, existindo lógica mais complicada no caso de uso. Da mesma forma que

para a camada de apresentação o teste de integração é necessário pelas mesmas razões. Testes de sistema, são todos importantes, dependendo do que o caso de uso fizer. Se houver algum tipo de atualização na base de dados devem, necessariamente, ser feitos teste de integridade.

6.8.1.3 Camada de dados

Para a camada de dados os testes de sistema são os mais importantes, principalmente o teste de carga e estresse associado com o de desempenho. Também o de integridade juntamente com a camada de negócio. Testes de unidade são raramente aplicados nesta camada a não ser que exista uma *stored procedure* complexa que justifique. Testes de aceitação também são pouco aplicados pois os requisitos já são testados nas duas primeiras camadas envolvendo esta. Teste de integração muitas vezes são necessários quando se utiliza *stored procedures.*

Com isso, pode-se enfatizar que em cada camada deste ambiente são necessários alguns níveis de teste. Parece óbvio, mas é importante atentar para este detalhe para que não sejam omitidos testes nas fases que seguem o projeto.

6.8.2 Sugestão de técnicas e critérios

Para implementar um caso de uso podem ser criados um ou mais componentes. Os critérios de teste a serem utilizados nos componentes que implementam o caso de uso, em cada estágio ou nível de teste dependem do grau de prioridade do caso de uso em questão, obtido no passo anterior desta estratégia.

Um componente relacionado a um caso de uso que, de acordo com a análise de risco e estabelecimento da prioridade, tem uma prioridade Alta, deve ser testado com critérios bem rigorosos, em todos os níveis em que estiver envolvido. Ou seja, se este caso de uso pode contemplar componentes da camada de apresentação, negócio e dados e, em todas estas camadas, para todos os níveis de testes (unidade, integração, sistema e aceitação), devem procurar utilizar os critérios mais rigorosos.

Com base nas prioridades, a Tabela 6.13 apresenta uma sugestão de testes e critérios que podem servir de base para selecionar os tipos de teste levando em consideração as características de cada caso de uso e devem ser analisadas juntamente com a sugestão de Mosley [45] (Seção 4.4).

TABELA 6.13 SUGESTÃO DE TESTES E CRITÉRIOS DE ACORDO COM A PRIORIDADE

Níveis de teste	Prioridade Alta	Prioridade Média	Prioridade Baixa
Teste de unidade	Teste estrutural Cobertura de Condição(2) Critério Todos Ramos (2) Teste funcional Análise do valor limite (2) Teste Baseado em erro Critério Análise de mutantes(2)	Teste estrutural Cobertura de Condição(2) Todos nós (2) Teste funcional Particionamento de equivalência (2)	Teste funcional Baseados em caso de uso (cenários) (2)
Teste de integração	Análise do valor limite (2) Teste de segurança (1,2,3) Teste de integridade (2,3)	Particionamento de equivalência (2)	Não necessário
Teste de sistema	Teste de carga (2,3) Teste de estresse(2,3) Teste de segurança(2,3) Teste de integridade(2,3) Teste de volume(3) Teste de recuperação de cópia de segurança (3) Teste de desempenho(1,2,3)	Teste de carga (2,3) Teste de segurança(2,3) Teste de integridade(2,3) Teste de desempenho(1,2,3)	Teste de segurança(2,3) Teste de integridade(2,3)
Teste de aceitação	Teste de usabilidade(1) Teste de ícones (1)	Teste de usabilidade(1)	Não necessário
Teste de regressão	Refazer todos os testes com os casos de usos que foram documentados para o módulo que foi alterado.	Refazer testes de unidade e sistema para verificação do impacto das alterações.	Refazer algum teste de sistema

NOTA: (1) representa Camada de Apresentação, (2) Negócio e (3) Dados

A ETACS estabelece algumas regras e orientações importantes:

1. Não deixar de realizar testes por causa do cronograma, ao invés disso, indicar que o projeto está atrasado.
2. O teste do sistema deve ser feito com o objetivo de encontrar um defeito, isso realmente é um ganho. Para que isso seja mais fácil, é recomendado sempre que possível que uma pessoa diferente da que desenvolveu projete e execute os testes.
3. Testes de desempenho são essenciais na arquitetura cliente-servidor e não devem ser omitidos nos casos de uso com prioridade Média e Alta. Com este tipo de teste é possível chegar a conclusões que estão faltando índice no banco, que a arquitetura utilizada não é a melhor para a característica do sistema, que tem um gargalo na rede, que micros estão mal configurados, etc.
4. Para auxiliar o registro dos dados de teste executados, é altamente recomendada a utilização de uma ferramenta. Caso não exista tal ferramenta, é importante que o desenvolvedor ou mesmo o testador com conhecimentos de programação, inclua comandos no programa para que todos os dados de entrada e respostas obtidas sejam gravados em um arquivo de texto. Pode ser aproveitado para gravar o tempo também

para auxiliar a análise do desempenho. Este arquivo será importante também para os testes de regressão. Este arquivo será muito útil quando o bom testador elaborar o plano de teste com casos de teste baseados em algum critério, ou seja, preparar dados de entrada coerentes com o objetivo de encontrar erro, colocando entrada e saída esperadas. Pois assim, juntando os dois arquivos, será possível utilizá-los na etapa de análise dos resultados, nos testes de regressão, na depuração e manutenção.

5. Outro detalhe importantíssimo é investir tempo com teste, deve-se prever em torno de 20% do tempo do projeto. A medida que isto venha sendo feito com critério os números positivos vão aparecer.
6. Documentar os erros encontrados em que data e versão, para registrar os ganhos obtidos com testes.
7. Devem-se realizar principalmente testes de integridade em casos de uso que façam atualização em banco de dados, principalmente quando mais de uma tabela é atualizada na mesma transação.

6.9 SÍNTESE DA ESTRATÉGIA ETACS

A Estratégia de Teste para Ambiente Cliente Servidor – ETACS tem o objetivo de simplificar a atividade e orientar os desenvolvedores nesta atividade tão importante. Para isso foram propostas etapas dentro do ciclo de vida iterativo de desenvolvimento, conforme esquematizadas na Figura 6.3.

A Tabela 6.14 e a Figura 6.3 mostram uma visão global e resumida da estratégia dentro das etapas de desenvolvimento do projeto conforme proposta no RUP (Seção 4.5.1), onde estas etapas acontecem para cada iteração. O que pode ser muito flexível, pois os sistemas mais simples podem ser implementados com apenas uma iteração.

TABELA 6.14 VISÃO GLOBAL DA ESTRATÉGIA ETACS

Fases do projeto	Ações da atividade de teste	Marco e Resultado final
Iniciação/ Etapa inicial	1. Levantar uma estimativa de custos com hardware, software e pessoas envolvidas, bem como a viabilidade da solução do problema, apresentar testes genéricos com base na descrição geral do sistema	Projeto aprovado/ Estimativa de custo de teste
Elaboração/ Levantamento de requisitos e definição da solução com modelos	1. Para cada caso de uso verificar coeficiente de prioridade 2. Com base na prioridade definir os tipos de testes 3. Fazer ou refazer o cronograma do projeto prevendo tempo para preparar e executar os testes que foram definidos. Deve-se prever tempo inicial maior para os dois primeiros casos de uso por causa da configuração do ambiente das máquinas. 4. Rever o plano de trabalho com relação a estimativa de custos 5. Elaborar casos de teste para testes de sistema (essencial) e	Análise concluída (do todo ou parte se abordagem espiral) Plano de teste completo

	regressão (se houver dados registrados) 6. Definir medidas de cobertura para os testes 7. Documentar	
Construção/ Análise/Projeto e Implementação	1. Elaborar casos de teste para executar testes de integração e unidade 2. Estabelecer medidas de cobertura 3. À medida que módulos ou componentes forem ficando prontos colocam-se em execução os casos de teste do projeto de casos de teste e se necessário geram-se mais para alcançar a cobertura determinada 4. Testes de integração são realizados 5. Na execução dos testes deve-se isolar as camadas e validar a arquitetura buscando melhor desempenho nas primeiras iterações 6. Documentar	Projeto de caso de teste
Transição/"Testes finais"	1. Execução dos testes de sistema, se houver manutenção, refazer os testes (teste de regressão) 2. Nos casos de prioridade alta, testar camada do servidor separadamente, cria-se um *stub* para acessá-lo com funcionalidade limitada e depois integra-o ao sistema e analisa-se o impacto. 3. Testes de validação ou aceitação 4. Documentar	Sistema ou parte dele pronto volta para elaboração
Transição/Implantação /Entrega – *Releases*	1. Execução de testes de aceitação, inicialização do sistema, configurações de máquina e testes do instalador 2. Teste beta 3. Documentar	Sistema entregue
Manutenção	4. Executar novamente todos os testes (teste de regressão), teste de unidade, integração e se necessário também de sistema, dependendo da prioridade e das mudanças.	Versão corrigida

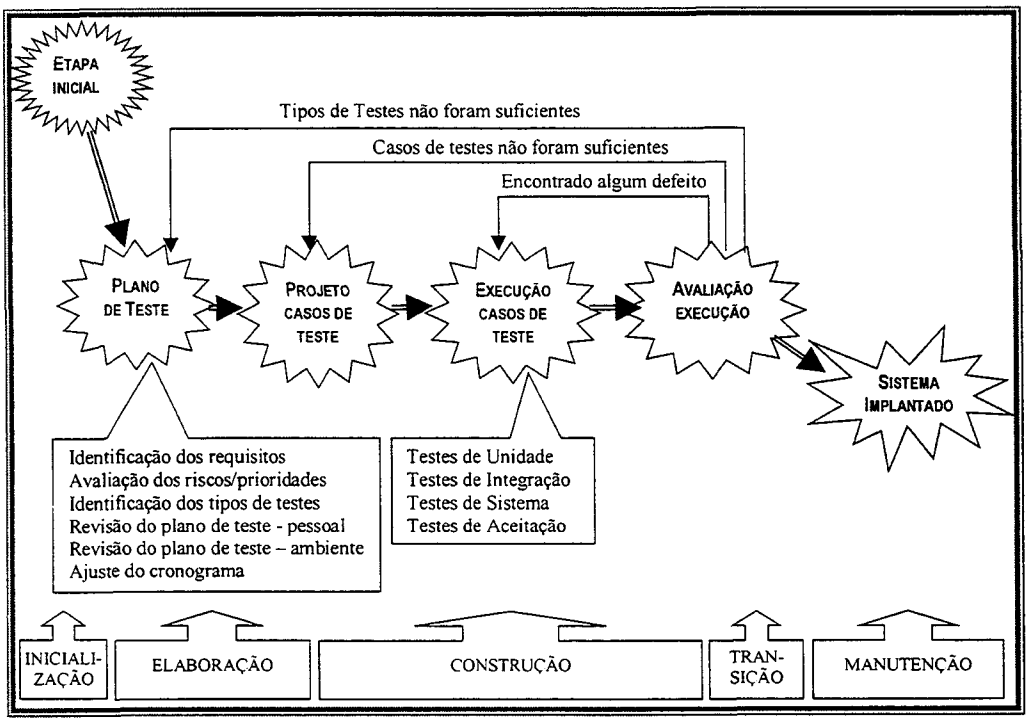


FIGURA 6.3 ESQUEMA DA ESTRATÉGIA ETACS DENTRO DAS FASES DO DESENVOLVIMENTO

6.10 CONSIDERAÇÕES FINAIS

Como foi identificado que o estabelecimento de prioridade proposto na ferramenta da Rational era bastante interessante e completo foi aplicado na ETACS alterando o estabelecimento dos conceitos conforme a Tabela 4.3, utilizada pela metodologia Test-Rx, ao invés da Tabela 4.4. Esta alteração foi feita com base nos estudos de Karlsson [30] que propôs a Tabela 6.3, pois foi verificado que é mais coerente e fácil de aplicar aos casos de uso segundo os conceitos.

Os testes propostos por Mosley foram integralmente aproveitados acrescentando apenas algumas orientações que foram considerados pertinentes. O *Gartner Group* considera em uma das fases do ciclo de vida do desenvolvimento com teste a inclusão de uma etapa para geração de *scripts* dos casos de testes projetados. A ETACS não considera isto uma etapa, apenas aconselha o uso de uma ferramenta para a execução dos testes que forem necessários.

O Pressman já tem trabalhado um pouco mais no ciclo de vida evolutivo de desenvolvimento mas definiu níveis de teste que podem ser adequados a qualquer ciclo e a ETACS procurou estabelecer estes níveis de teste para o ciclo iterativo apresentado pelo RUP, no qual a ETACS foi baseada.

A estratégia de teste proposta tinha o objetivo de sugerir tipos de critérios e técnicas de testes baseados na prioridade dos casos de uso, mas levar em consideração somente as prioridades é impraticável, pois os tipos de testes dependem diretamente dos requisitos e funcionalidade. Para um caso de uso com muita consulta e nenhuma atualização, testes de integridade não são necessários e testes de desempenho são altamente recomendados, e componentes que tenham pouca lógica, apenas chamadas, testes de integração são imprescindíveis.

A arquitetura cliente-servidor interfere diretamente na etapa de execução dos casos de teste e está fortemente dependente dos testes de sistema que influenciam na definição dela.

Os critérios de teste tradicionais com técnica estrutural e funcional podem ser aplicados igualmente considerando um módulo ou um componente.

A ETACS orienta o desenvolvedor durante todo o processo para selecionar os tipos de testes e quais testes devem ser aplicados para que o sistema seja construído com uma arquitetura mais adequada para as características do mesmo. A ETACS enfatiza a validação da arquitetura para que o risco do sistema de não atender ao requisito de desempenho diminua e o custo de alterar esta arquitetura seja o mínimo possível.

A ETACS auxilia na simulação do ambiente do cliente apontando detalhes que aproximam as diferenças e também orienta os testes de desempenho resolvendo os dois principais problemas encontrados na pesquisa realizada, desempenho e simulação de ambiente.

A estratégia orienta os testadores a documentar cada fase, quando não se utiliza uma ferramenta para automatizar os testes, muito dessa documentação é manual, mas necessária para a realização dos testes de regressão e avaliação dos ganhos com a aplicação dos testes.

7 ESTUDO DE CASO

Este capítulo apresenta resultados da aplicação da ETACS em um caso real de uma empresa. Utiliza-se a abordagem incremental de desenvolvimento (Seção 4.5) e casos de uso para a especificação de requisitos. Como o projeto é bastante extenso, foi utilizado apenas um módulo a fim de validar esta estratégia de teste, e fazer ajustes ou recomendações com base nos resultados positivos ou negativos obtidos. A seguir está descrito o que foi realizado em cada uma das etapas da ETACS com suas ponderações no que diz respeito ao que foi descoberto e o que se deve gerar a partir de cada etapa.

7.1 ETAPA INICIAL

Nesta etapa inicial não serão abordadas apenas as informações úteis para a atividade de teste, e sim o que resulta desta etapa é analisado para gerar os casos de uso iniciais que serão analisados no decorrer da atividade. A idéia não é também fazer uma documentação completa do sistema pois isso não é o foco. O que se pretende é mostrar informações poderiam ser levantadas em cada etapa, principalmente as que servem de base para os casos de uso selecionados para o teste.

7.1.1 O que fazer

Conforme orienta a estratégia ETACS, na etapa inicial é preciso levantar várias informações do projeto, com estas informações é possível estimar inicialmente o custo dos testes. Não existe uma lista rigorosa a ser seguida, todas as informações conseguidas podem ser importantes.

7.1.2 Resultado final desta etapa

Escopo

O software fará parte de um sistema Y, desenvolvido na plataforma Natural/Adabas, que existe e está em funcionamento. O novo software será tratado, a partir de agora, como

software X. O novo software trata do controle financeiro de arrecadações e multas, originadas dos autos de infrações do Código de Trânsito Brasileiro e permite uma visão restrita dos dados para os órgãos.

Descrição resumida do sistema

Existe um sistema em mainframe que gera a distribuição do que foi arrecadado com cada multa para todas as entidades envolvidas na infração e arrecadação da mesma. Como cada órgão que recebe o valor pode ter que repassar para órgãos subordinados, isto gera uma árvore. Esta árvore de distribuição do pagamento das multas é enviado diariamente para o Software X, que está sendo apresentado neste estudo de caso.

Este sistema permitirá que os órgão envolvidos nas distribuições possam consultar as multas que já foram pagas e falta o órgão fazer o repasse, os valores que foram repassados e diversas formas diferentes de consultar e conforme as restrições da sua posição hierárquica na árvore.

Tecnologia adotada

Quanto à técnica de análise, definiu-se pela continuidade no uso da Análise Orientada a Objetos e o processo unificado de desenvolvimento de software [40]. Este projeto apresenta muitas exceções, o que poderia dificultar a manutenção.

Quanto à tecnologia, são propostas funcionalidades mais versáteis (interface gráfica) e possibilidade de integração de aplicativos de mercado à solução, como editores de texto, planilhas eletrônicas, etc.

A arquitetura da solução é mais flexível do que a existente no mainframe, pois permite que a mesma transação seja executada de forma direta (sincronizada) ou indireta, neste último caso usando filas de requisições. Os componentes do sistema são móveis, ou seja, podem mudar de ambiente de execução à medida que as necessidades de recursos se alterem, sem precisar modificar o código da aplicação. A mesma arquitetura de aplicação pode ser utilizada em uma máquina *stand-alone*, em uma pequena rede ou em um ambiente com alto grau de distribuição.

Infra-estrutura para desenvolvimento

- *Hardware* - O ideal é que as estações de trabalho tenham a seguinte configuração mínima, que permitirá uma boa produtividade e desempenho do software: Pentium 450 Mhz com 64 Mb de memória e 3 GB de HD. Para a documentação do projeto poderá ser utilizada área do servidor da rede local do setor responsável.

- Software - Visual Studio Interprise 6.0; Case da Rational Rose; Planilha eletrônica: Excel; Editor de texto: Word 97 (para documentação).
- *Treinamento* - Interdev (Rubens, Roberto, Giovana e Lisiane); SQL Server (Rubens, Roberto, Giovana e Lisiane); Análise Orientada a Objetos (Rubens, Roberto, Giovana, Jumara e Regina).

Etapas/produtos a serem gerados

- especificação de Requisitos atualizada;
- diagrama de classes atualizado;
- diagrama de Transição de Estado (DTE) dos objetos, quando for relevante para a discussão e o entendimento;
- descrição resumida dos casos de uso, destacando os eventos que disparam o caso de uso;
- descrição detalhada dos casos de uso para subsidiar a identificação e especificação dos métodos das classes;
- diagrama de classes acrescido dos métodos (visão estática), identificando as classes que pertencem a cada uma das camadas;
- diagrama de seqüência, que representa a dinâmica dos casos de uso;
- especificação da interface (fronteira);
- plano de testes;
- bases de dados para teste;
- códigos fonte e componentes;
- aplicação em ambiente de produção.

Complexidade

Projeto de grande complexidade, com destaque para os seguintes fatores:

a) Diversidade e a própria complexidade das regras de negócio;

b) Nº de agentes envolvidos;

c) Necessidades diferentes dos clientes, devendo ter solução padrão que atenda a todos;

d) Forte integração com rotinas, procedimentos e sistemas já existentes.

Arquiteturas

As arquiteturas possíveis para o sistema Y podem ser vistas na Tabela 7.1.

TABELA 7.1 ARQUITETURA DO NOVO SOFTWARE

ID	Descrição		
1	Terminais		Mainframe
	• Terminais burros de mainframe ou • PCs emulando terminais		• Servidor de apresentação (formato caracter) • Servidor de aplicação (transações Natural em ambiente CICS) • Servidor de dados (Adabas)
2	Servidor de Apresentação	*Gateway*	*Mainframe*
	• Interface gráfica em PCs (*thin client*)	• Monitor de transação em plataforma NT, fazendo a integração com o CICS no mainframe	• Servidor de aplicação (transações Natural em ambiente CICS) • Servidor de dados (Adabas)
3	Servidor de Apresentação	Servidor de Aplicação	Servidor de Dados
	• Interface gráfica em PCs (*thin client*)	• Monitor de transação em plataforma NT, executando as transações do sistema (casos de uso) e acessando o Adabas via CICS	• Mainframe (CICS – Natural – Adabas)
4	Servidor de Apresentação	Servidor de Aplicação	Servidor de Dados
	• Interface gráfica em PCs (*thin client*)	• Monitor de transação em plataforma NT, executando as transações do sistema (casos de uso) e acessando o Adabas via CICS	• Mainframe (CICS – Natural – Adabas)
	• Interface HTML (funções de consulta relacionadas ao Controle de Crédito e ao Controle do Repasse)	• *Servidor Web*, contendo URLs que apontam para transações hospedadas no servidor de aplicação (monitor de transação)	

Volumes / Sazonalidade

Não foi feito um levantamento detalhado sobre volumes, simplesmente foi calculado o número de registros e a quantidade de bytes ocupados pelas entidades que consomem mais espaço, no órgão que movimenta a maior quantidade de informações e, em seguida, foram feitas projeções empíricas usando esses números básicos. Isso significa que o cálculo mostra a ordem de grandeza da base de dados do sistema, mas não um número exato.

Considerando-se conforme a Tabela 7.2, a implantação do sistema em 7 órgãos como o XYZ e a manutenção das informações na base de dados por 24 meses, a conclusão a que se chega é que a entidade Pagamento necessita de 252 Mb (1,5 Mb x 7 órgãos x 24 meses), isto é, 0,246 Gb.

Outra entidade importante para o cálculo é DistribuiçãoPagamento. Foi estimada uma média de 4 registros de DistribuiçãoPagamento para cada ocorrência de Pagamento. Já que o tamanho do registro de DistribuiçãoPagamento é 125 bytes, o espaço necessário para armazenar essa informação é: 4 x 12.898 (Pagamentos no mês) x 7 (órgãos) x 125

(tamanho do registro) x 24 (prazo de disponibilidade da informação) = 1.033,24 Mb = 1,009 Gb.

Espaço total necessário para as duas entidades mais críticas: 1,009 + 0,246 = 1,255 Gb. Para o sistema inteiro, acredita-se que 3 Gb é mais do que suficiente. Quando for necessário estimar número de transações (número de execuções de cada caso de uso), pode-se tomar por base a quantidade de documentos gerados por mês.

TABELA 7.2 VOLUME DE DADOS PREVISTO

Órgão	Período	Entidade	Tamanho Registro	Tipo Docto	Qtd de Doctos	Espaço Necessário
XYZ	06/99	Pagamento	116 bytes	GRLAV	4069	460,0 Kb
				GRD	30	3,4 Kb
				GRT	5573	631,0 Kb
				GRM	3226	365,4 Kb
Total 1.459,8 Kb = 1,43 Mb ~ 1,5 Mb em um mês						

Forma de documentação

Utilização da ferramenta case *Rational Rose* para modelagem e geração dos gráficos e figuras, mas a documentação dos modelos deve ser feita na sua forma final através do *Word*. Esta duplicação de documentação se fez necessária devido ao fato da empresa não ter disponível ainda o módulo de formatação da documentação oferecido por esta ferramenta *case*. Os documentos gerados devem ser arquivados em um diretório da rede local pois já possui mecanismos diários de *backup*.

7.2 PLANO DE TESTE

Nesta etapa, a estratégia ETACS define os passos e quais os itens que devem ser documentados para que se faça um bom plano de teste. Serão seguidos os passos apresentados na Seção 6.2. Esses 6 passos são identificação dos requisitos, avaliação dos riscos e definição de prioridades, identificação dos tipos de testes, técnicas e critérios, revisão do plano de trabalho quanto aos recursos humanos e ambiente e ajuste do cronograma.

A seguir é apresentado o que resulta de cada um desses passos, apresentando o resultado final seguindo os roteiros propostos e explicativos da ETACS.

7.2.1 Identificação dos requisitos

Requisitos funcionais

A Figura 7.1 apresenta o diagrama de casos de uso de parte do módulo deste escopo do Software X. São 8 os casos de uso utilizados. Eles encontram-se detalhados no ANEXO 1.

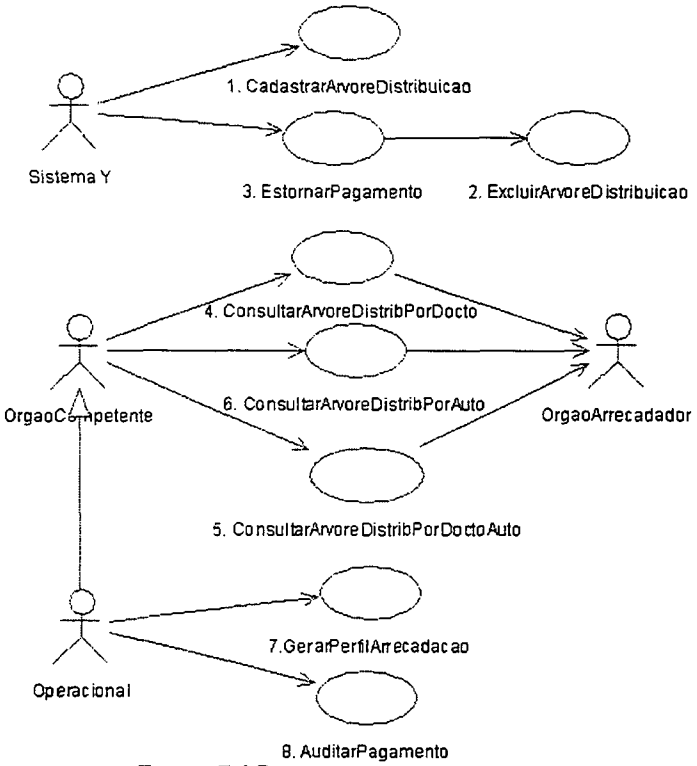


FIGURA 7.1 DIAGRAMA DOS CASOS DE USO

Requisitos não funcionais

- Desempenho - Tempo aceitável para as consultas simples com poucos dados (até 500 bytes) deve ser 2 a 3 segundos, o que corresponde no sistema a uma árvore com 5 distribuições.
- Segurança - O sistema deve bloquear a execução de métodos por pessoas não autorizadas, nem permitir a entrada direta no banco de dados garantindo a segurança no banco. A senha não deve ficar explícita em nenhum código de programação.
- Precisão - Como muitos dos valores são armazenados em UFIR, os valores devem ser armazenados e calculados com 4 dígitos decimais embora a formatação para o usuário deve ser com 2 dígitos na casa decimal.

7.2.2 Avaliação dos riscos e prioridades

O segundo passo do Plano de Teste especificado pela ETACS é identificar o grau de prioridade analisando os riscos de cada caso de uso identificado, para se estabelecer os critérios e tipos de teste. Segundo o roteiro proposto para esta tarefa (Seção 6.7), é necessário avaliar e justificar estes casos de uso segundo seis itens, conforme descritos a seguir. As justificativas para cada item se encontram no APÊNDICE C e o agrupamento destes conceitos conforme orienta a ETACS estão unificados na Tabela 7.3.

TABELA 7.3 AGRUPAMENTO DOS CONCEITOS DE TODOS OS ITENS

Descrição do Caso de uso	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
1. Cadastrar Distribuição das multas	Crítico	Crítico	Pouca importância	Prioritário	Pouca importância	Crítico
2. Excluir a Distribuição	Crítico	Regular	Regular	Regular	Pouca importância	Pouca importância
3. Estornar Pagamento	Prioritário	Regular	Regular	Pouca importância	Pouca importância	Pouca importância
4. Consultar Árvore de Distribuição por Docto	Prioritário	Crítico	Prioritário	Crítico	Prioritário	Prioritário
5. Consultar Distribuição Docto-Auto	Pouca importância	Prioritário	Prioritário	Prioritário	Prioritário	Prioritário
6. Consultar Distribuição por Auto	Pouca importância	Prioritário	Prioritário	Crítico	Prioritário	Prioritário
7. Gerar Perfil de Arrecadação	Prioritário	Regular	Regular	Regular	Regular	Regular
8. Auditar Pagamento sem Crédito	Regular	Pouca importância	Exceção de qualidade	Exceção de qualidade	Exceção de qualidade	Exceção de qualidade

Com base na Tabela 7.3 são feitas médias, multiplicações por pesos e identificação da faixa que se encontra o somatório total para chegar a um conceito final e classificando-se por ordem de prioridade resulta na Tabela 7.4. Esta tarefa foi programada por uma planilha eletrônica. Para estes casos de usos o detalhamento de como foi aplicado passo a passo o roteiro para a elaboração das tabelas intermediárias está descrito no APÊNDICE D.

7.2.3 *Identificação dos tipos de testes, técnicas e critérios*

Para apresentar como foi aplicado este passo é utilizado o Caso de Uso 4 - Consultar Árvore de Distribuição por Docto, por ter prioridade alta e o maior coeficiente representado pelo somatório, conforme apresenta a Tabela 7.4. E também o Caso de Uso 1 – Cadastrar árvore de distribuição pois efetua uma atualização na base.

Analisando a Tabela 7.4 e os requisitos do usuário levantados em casos de uso (ANEXO 1), é possível considerar quais testes serão pertinentes a cada caso de uso e pode ser utilizada a sugestão fornecida pela ETACS na Tabela 6.13.

TABELA 7.4 RESULTADO FINAL COM AVALIAÇÃO DO GRAU DE PRIORIDADE

Descrição do Caso de uso	Item 1,2,3 peso=3	Item 4 Peso=2	Item 5 Peso=1	Item 6 peso=3	Somatório	Conceito Final
4. Consultar Árvore de Distribuição por Docto	14,00	10,00	4,00	12,00	40,00	ALTA
1. Cadastrar Distribuição das multas	12,00	8,00	2,00	15,00	37,00	ALTA
6. Consultar Distribuição por Auto	10,00	10,00	4,00	12,00	36,00	ALTA
5. Consultar Distribuição Docto-Auto	10,00	8,00	4,00	12,00	34,00	ALTA
7. Gerar Perfil de Arrecadação	10,00	6,00	3,00	9,00	28,00	MÉDIA
2. Excluir a Distribuição	11,00	6,00	2,00	6,00	25,00	MÉDIA
3. Estornar Pagamento	10,00	4,00	2,00	6,00	22,00	MÉDIA
8. Auditar Pagamento sem Crédito	6,00	2,00	1,00	3,00	12,00	BAIXA

Os testes que não se aplicam para o caso de uso devem ser eliminados, neste Caso de Uso

4, o teste de integridade, pois este caso de uso é apenas uma consulta. E pode ser incluído algum teste que se faça necessário por causa de alguma restrição no caso de uso.

Existem casos de usos que não têm código, só chamadas de funções, e para estes casos, não faz sentido o teste de unidade, mesmo com prioridade Alta, já, um teste de integração é importante para verificar as chamadas das funções e passagem de parâmetros.

Fazendo estas ponderações , os testes considerados pertinentes a este caso de uso estão apresentados na Tabela 7.5.

TABELA 7.5 IDENTIFICAÇÃO DE TESTES PARA O CASO DE USO 4

Níveis de teste	Prioridade ALTA
Teste de unidade	Teste estrutural Cobertura de Condição Critério Todos Ramos Teste funcional Análise do valor limite
Teste de integração	Teste de segurança (1,2,3)
Teste de sistema	Teste de carga (2,3) Teste de estresse(2,3) Teste de segurança(2,3) Teste de volume(3) Teste de desempenho(1,2,3)
Teste de aceitação	Teste de usabilidade(1)
Teste de regressão	Refazer todos os testes com os casos de usos que foram documentados para o módulo que foi alterado.

Outro exemplo utilizando um caso de uso que faça atualização é o segundo na lista de prioridades. Da mesma forma foram identificados os testes para o Caso de Uso 1 – Cadastrar Árvore de Distribuição, conforme a Tabela 7.6.

TABELA 7.6 IDENTIFICAÇÃO DOS TESTES PARA O CASO DE USO 1

Níveis de teste	Prioridade ALTA
Teste de unidade	Teste estrutural Cobertura de Condição Critério Todos Nós Teste funcional Análise do valor limite
Teste de integração	Teste de segurança (1,2,3) Teste de integridade (2,3) Teste de sabotagem (2,3)
Teste de sistema	Teste de carga (2,3) Teste de estresse(2,3) Teste de segurança(2,3) Teste de integridade(2,3) Teste de volume(3) Teste de desempenho(1,2,3)
Teste de aceitação	Não tem iteração com o usuário, basta que os dados estejam corretamente cadastrados
Teste de regressão	Os dados não podem ser reutilizados para o cadastramento pois não é permitida a duplicação.

7.2.4 Revisão do plano de trabalho quanto aos recursos humanos

Para realizar os testes é necessário uma equipe, idealmente pessoas diferentes das responsáveis pelo desenvolvimento deste sistema. Para estes casos de uso um analista do negócio, projetou os casos de teste e outro analista responsável pela execução dos testes, registros e ajustes nos programas.

7.2.5 Revisão do plano de trabalho quanto ao ambiente

O resultado deste passo deve ser orientações práticas que devem ser executadas ou que deve ser documentado relativos ao ambiente de teste. Um exemplo de como pode ser estas orientações está demonstrado a seguir.

Analisando o ambiente de vários micros constatou-se que será necessário preparar várias máquinas que tenham acesso ao servidor onde está o *COM Service (MTS)*, tenham permissão para instalação do *proxy* do componente gerado (alguém com direitos de administrator na máquina), tem que ter o software *Windows Installer* na máquina para registrar o componente (ou deve ser instalado esse programa), requer a versão 5.0 ou superior do *Internet Explorer*, *MDAC* versão 2.6.

Para executar os testes de estresse, carga e desempenho são necessários no mínimo seis máquinas rodando simultaneamente com esta configuração. Com base no que foi definido para a execução dos casos de teste na ETACS na Seção 6.4.

Para verificar a integridade, é necessário que em um dos micros esteja instalado uma ferramenta que tenha acesso ao servidor de banco de dados *Sql Server*, *Query Analyser* e um *login* de acesso válido.

7.2.6 Ajuste do cronograma

O cronograma deve ser revisto incluindo o tempo para a realização dos testes em cada etapa. Como demonstração da aplicação deste passo pode-se definir:

1. Para a realização dos testes de integridade e segurança considerar 3 horas para cada caso de uso
2. Para testes de carga, volume, estresse e desempenho, considerar 12 horas para o primeiro caso de uso e 4 horas para os demais
3. Para os testes de unidade e integração, considerar 6 horas para cada caso de uso

7.3 PROJETO DE CASOS DE TESTE

Os tipos de testes que foram identificados na Seção 7.2.3 para cada caso de uso devem ser utilizados para preparar os casos de teste definindo ou revendo a cobertura (critério de satisfação/parada do teste para este caso de uso). A revisão da cobertura para os testes identificados é apresentada na Tabela 7.7. Os valores apresentados nesta tabela devem ser definidos pelo gerente do teste caso o tipo de teste escolhido não defina um critério de parada.

Foram projetados os dados de teste e apresentados parte deles no APÊNDICE F para cada tipo de teste identificado na seção anterior. Foram projetados casos de teste ou detalhadas instruções quanto aos dados para execução dos testes de condição, de segurança, estresse, de desempenho *e* de integridade para os casos de uso utilizados como exemplo da aplicação da estratégia.

TABELA 7.7 TIPOS DE TESTES COM CRITÉRIO DE PARADA

Tipos de Teste	Critério de parada
Cobertura de Condição	2 condições válidas e 2 inválidas
Critério Todos Ramos	Todos os ramos da função serem exercitados
Análise do valor limite	Limite, imediatamente abaixo do limite e imediatamente acima dele
Teste de segurança	Validar se um usuário executa componentes que não tem direito, se acessa direto o banco e casos que tenha direito. Trocando senha para validar se há verificação.
Teste de carga	Entrar com uma carga de 10.000 registros
Teste de estresse	10 usuários no mínimo simultâneos acessar os componentes automaticamente por 1 hora e meia direto
Teste de volume	Consulta de um período de 1 mês
Teste de desempenho	1 segundo para documentos com até 7 distribuições e no máximo 5 segundos para documentos com mais de 7 distribuições.
Teste de usabilidade	Utilizar *check list* [44]

7.4 EXECUÇÃO DOS CASOS DE TESTE

Nesta fase devem ser executados os casos de teste conforme as observações feitas na etapa anterior de projeto de casos de teste e registrados os resultados obtidos para que se possa fazer uma análise dos resultados.

No APÊNDICE G são mostrados os resultados obtidos a partir dos casos de testes propostos para os casos de uso em questão e no APÊNDICE H são apresentados os diagramas de componentes para as varias arquiteturas do ambiente cliente servidor utilizadas nos testes.

Para execução do teste de desempenho foi construído a partir de um programa em *visual basic*, chamadas contínuas a componentes e marcando tempo, porque não havia

disponibilidade de ferramenta. O programa toma por base uma entrada de dados em arquivo texto de 10.740 linhas e seleciona aleatoriamente um número múltiplo de 500 para ser o ponto de partida. Também controla o tempo para que possa ser programado um horário de início. Para que isso funcione todos os micros devem estar com o horário sincronizado com um mesmo servidor. O programa também chama aleatoriamente um dos 3 casos de usos, ou seja, este teste está englobando Caso de Uso 2 (F109), Caso de Uso 3 (F105) e Caso de Uso 4 (F106).

Conforme observado no APÊNDICE G, é calculado o tempo de chamada das funções e outro tempo considerando a apresentação do resultado em tela inclusive; também é calculado o número de distribuições que retorna para cada consulta e também o número de bytes que retornou. Os resultados dos tempos são gravados em arquivo texto, e posteriormente, são agrupados por função, por número de máquinas que executaram o teste. A análise mais justa é feita considerando a mesma máquina, pois desconsidera aspectos de ambiente da máquina específica.

7.5 AVALIAÇÃO DOS CASOS DE TESTE

Pode ser observado que alguns resultados não foram iguais ao que se esperava. E para alguns testes como o caso do teste de segurança, foram identificadas necessidades adicionais de configuração para que o caso de uso pudesse ser executado e testado. No APÊNDICE G, estão apresentados mais dados sobre as execuções dos casos de teste para permitir as avaliações mostradas a seguir.

7.5.1 Teste de condição

A Tabela 7.8 apresenta o resultado da execução do caso de teste 6, que revela a presença de um defeito pois existe diferença entre o resultado obtido e esperado. Os casos de teste 3, 5 e 6 revelaram defeito no programa. Os casos de teste 1 a 5 são listados no APÊNDICE G.

TABELA 7.8 CASO DE TESTE 6 PARA CASO DE USO 4 – (F106)

Nome do dado	Dado de entrada	Saída esperada	Resultado obtido
Id Docto:	2000632046716011	Auto=Distribuições: **275350W000220642**=8709,8710,8711 **275350W000277105**=8722,8723,8724 **275350W000401196**=8729,8730,8731	Auto=Distribuições: **16100E000547813**=8719,8720,8721 **275350W000220642**=8709,8710,8711,8712,8713,8714,8715 **275350W000277105**=8722,8723,8724,8725,8726,8727,8728 **275350W000401196**=8729,8730,8731,8732,8733,8734,8735
Autenticação:	0		
Tipo do Docto:	2		
Período Crédito:	18/05/2001		
Período Pagamento:	Nulo		
Id do órgão que consulta:	275350		

7.5.2 *Teste de estresse*

Quando mais de um usuário estava acessando, algumas máquinas apresentaram um tipo de erro: "*3704 – Operation is not allowed when the object is closed*" enquanto outras (mais de uma) estavam acessando normalmente os mesmos componentes. É necessário encontrar a razão do erro, pode ser uma falha no servidor da camada de negócio que não consegue gerenciar muitos usuários.

7.5.3 *Teste de desempenho*

O sistema foi projetado e implementado considerando as melhores práticas para desenvolvimento de componentes neste tipo de arquitetura baseados nos conceitos abordados em [39],[49] que ressalta a importância da arquitetura no ambiente cliente-servidor. O APÊNDICE H apresenta o diagrama de componentes esquematizados para as diferentes arquiteturas avaliadas. Com base nestes conceitos é que foi projetada uma arquitetura inicial, em que são implementados os conceitos de transação, segurança, desempenho (*threads*), usabilidade.

Esta primeira arquitetura é referida nos exemplos com a sigla CS3D e, em resumo, tem 3 pacotes que ficam na camada de negócio, um com várias DLLs, uma para cada caso de uso. Cada DLL cria um componente de um outro pacote que executa no mesmo processo, este segundo pacote contém os componentes de uso geral. Este segundo pacote cria um outro componente de um terceiro pacote (processo diferente) que acessa os dados no SGBD. Este terceiro pacote contém uma DLL para acessar cada uma das tabelas.

Mais detalhes das diferentes arquiteturas podem ser encontrados em [39], o importante aqui é saber que alterando alguns detalhes, como colocando vários componentes (classe que pode ser instanciada em um objeto) em uma DLL, ou cada componente em uma DLL separada, separar em vários pacotes ou apenas um com tudo dentro, alterar a interface dos métodos para um tipo *result set, array* de *strings* ou um tipo *string*, tudo isso altera a arquitetura do sistema e interfere no tempo final para o usuário. O importante é que não existe uma arquitetura ideal para tudo e sim depende de cada sistema.

Como pode ser observado na Tabela 7.9, a média do tempo que os componentes levam para executar a função para uma máquina é bastante aceitável e não muda muito quando tem 11 usuários executando o componente. Isto pode ser verificado para todas as 4 máquinas com dados registrados no APÊNDICE G.

Problema identificado:

Algo chamou a atenção: o tempo que leva para mostrar os resultados na tela foi muito além do esperado e muito mais alto que o tempo de executar o componente e trazer os dados para a camada do cliente. Demonstrando algum problema na lógica dessa apresentação, algum componente ou software não está bem configurado ou algo a ser descoberto. É intrigante o fato de uma média de 5 registros levar 1,2 segundos para ser trazido da base de dados e levar 9 segundos, em média, para mostrar esses dados na tela.

TABELA 7.9 TESTE DE DESEMPENHO PARA 1ª ARQUITETURA NA MAQ07ROSELI

Maq	QtdMq	Função	CS3-D			
			Bytes	Regs	TpCmp	TpTran
07Roseli	1	F105	489,75	4,25	2,819	8,280
		F106	1113,14	9,71	1,725	15,519
		F109	503	4,43	2,970	8,771
		Todas	692,32	3510,5	2,519	10,74
	11	F105	553,94	4,96	2,540	10,773
		F106	546,75	4,91	1,678	9,305
		F109	553,30	4,96	1,264	9,330
		Todas	551,35	7759,5	1,805	9,780

Origem do problema encontrada:

Para tentar identificar a origem do problema foi construída uma aplicação para acesso direto ao banco sem passar pelo componente (associada à sigla CS2) que corresponde a uma arquitetura cliente-servidor duas camadas e outra arquitetura juntando todos os componentes em uma única DLL que roda em um pacote no monitor de transação (CS3M). Mais uma modificação foi feita, agora na interface do componente, o tipo do retorno não mais um *recordset* e sim um *array* de *string* e associou-se com a sigla CS3Mar.

Um dos resultados deste comparativo pode ser visto na Tabela 7.10 e Figura 7.2.

TABELA 7.10 TEMPO DE COMPONENTE PARA 1 E 8 MÁQUINAS

F106-1	Contador	Média	Mediana	CV	Indicativo	Percentil 75	Percentil 90
CS2	352	1,55	1,20	140,49	1,20	1,56	2,11
CS3M	90	1,44	1,12	137,22	1,12	1,65	1,86
CS3MAr	463	0,87	0,62	126,56	0,62	1,08	1,19
CS3D	88	1,28	1,21	46,50	1,21	1,81	1,93
F106-8	**Contador**	**Média**	**Mediana**	**CV**	**Indicativo**	**Percentil 75**	**Percentil 90**
CS2-8	114	3,67	3,44	42,60	3,44	4,40	5,79
CS3M-8	72	1,90	1,58	69,98	1,58	2,27	3,20
CS3MAr-8	239	1,10	0,82	75,53	0,82	1,20	1,88
CS3D-8	90	1,73	1,50	61,63	1,50	2,10	2,66

Foi procurada a medida de tendência central mais adequada para descrever a população através da média e mediana avaliando o coeficiente de variação. Outro dado interessante é o percentil que apresenta o percentual de valores que está abaixo do calculado. Pode ser observado através desta tabela que o tempo que o componente leva para executar e trazer

o resultado pela rede, chamado tempo de componente, é bastante semelhante para CS2, CS3M e CS3D, quando sendo executado por uma máquina.

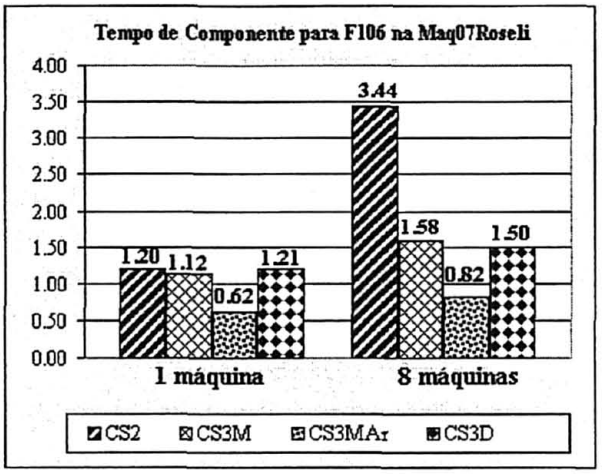


FIGURA 7.2 TEMPO DE COMPONENTE PARA MAQ07ROSELI

Se forem colocadas 8 máquinas, como é o caso no exemplo, estes valores aumentam em diferentes proporções. Ou seja a escalabilidade afeta bastante a arquitetura CS-2, pois para um usuário leva em torno de 1,2 segundos e para 8 usuários leva 3,44 segundos, enquanto para as outras arquiteturas estes valores aumentaram um pouco, mas não chegaram perto do dobro.

Observou-se que o comportamento foi semelhante nas várias máquinas que rodaram os testes. A quantidade de dados que foi passada pela rede não influenciou nestes valores pois foram muito semelhantes como pode ser observado, Tabela 7.11.

TABELA 7.11 NÚMERO DE BYTES PARA 1 E 8 MÁQUINAS

F106-1	Contador	Média	Mediana	CV	Indicativo	Percentil 75	Percentil 90
CS2	352	531,66	557,00	9,49	532	557	559
CS3M	90	533,11	557,00	8,51	533	557	557
CS3MAr	463	601,06	557,00	27,33	601	562	567
CS3D	88	532,91	557,00	8,38	533	557	557
F106-8	**Contador**	**Média**	**Mediana**	**CV**	**Indicativo**	**Percentil 75**	**Percentil 90**
CS2-8	114	1133,76	568,00	114,55	568	1124	3549
CS3M-8	72	716,31	463,00	89,90	463	705	1709
CS3MAr-8	239	3837,78	2639,00	96,16	2639	9349	9387
CS3D-8	90	616,54	694,50	17,28	617	700	705

O tempo total para executar o componente e apresentar os dados retornados para o usuário, ou seja incluindo tempo de interface foi calculado separado, e chamado tempo de transação, apresentado na Tabela 7.12.

Como pode ser observado pelas Tabela 7.10 e Tabela 7.12, o tempo que a arquitetura CS2 leva para executar o componente para um usuário é 1,2 segundos e o tempo de transação, incluindo mostrar na interface, é 1,26. Já na arquitetura CS3D, de 1,21 passou para 6,59. Comportamento semelhante para a arquitetura CS3M que de 1,12 passou para 6,52. Com isso foi realizada mais uma alteração na arquitetura mudando a interface de retorno desses

métodos. E o resultado foi que para esta nova arquitetura CS3Mar de 0,62 passou para 0,64. Com isso pode-se constatar que a arquitetura influencia muito o desempenho dos componentes e são imprescindíveis estas avaliações.

TABELA 7.12 TEMPO DE TRANSAÇÃO PARA 1 E 8 MÁQUINAS

F106-1	Contador	Média	Mediana	CV	Indicativo	Percentil 75	Percentil 90
CS2	352	1,61	1,26	136,24	1,26	1,61	2,16
CS3M	90	7,24	6,52	42,89	6,52	7,03	7,93
CS3MAr	463	0,90	0,64	124,00	0,64	1,10	1,21
CS3D	88	6,59	6,46	26,21	6,59	6,98	7,55
F106-8	**Contador**	**Média**	**Mediana**	**CV**	**Indicativo**	**Percentil 75**	**Percentil 90**
CS2-8	114	3,74	3,49	42,46	3,49	4,53	5,93
CS3M-8	72	10,86	7,98	79,94	7,98	11,17	24,21
CS3MAr-8	239	1,17	0,89	72,46	0,89	1,27	1,96
CS3D-8	90	8,84	8,57	24,88	8,84	9,50	11,10

Outro detalhe observado nas amostras é que embora os tempos dos componentes para as arquiteturas 3 camadas sejam baixos, a primeira vez em que são executados, leva mais de 10 vezes este tempo. Avaliando este detalhe surgiu a necessidade de testar as arquiteturas baseadas em componentes em intervalos de tempo, uma vez que a característica mais comum dos sistemas corporativos não são sistemas sob estresse. Para simular este ambiente foram executados testes com as arquiteturas CS3MAr aguardando 3 minutos (CS3Mar-T3) e aguardando 10 minutos (CS3Mar-T10) entre as chamadas. A interface da arquitetura CS3D foi alterada e submetida a testes em intervalos de tempo. O resultado na máquina Maq03Dante para o Caso de Uso 4 (F106) está apresentado na Figura 7.3.

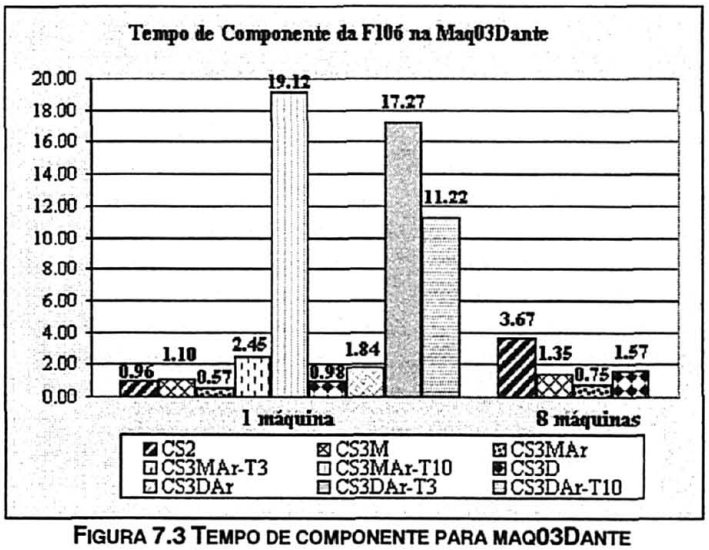


FIGURA 7.3 TEMPO DE COMPONENTE PARA MAQ03DANTE

Dentre várias informações que podem ser concluídas analisando as Figura 7.3 e Figura 7.4, a que foi procurada comparar foi que as arquiteturas CS3DAR e CS3MAR quando submetidas ao estresse têm um comportamento com tempo de resposta excelente. Contudo, quando são executadas em intervalos de tempo de 3 minutos ou 10 minutos, estes valores crescem demais.

O que pode ser concluído para a estratégia com estes resultados é que quando arquiteturas cliente-servidor 3 camadas estiverem sendo testadas para avaliar desempenho, deve-se avaliá-las sob estresse e em intervalos de tempo, pois isso influencia o tempo de execução.

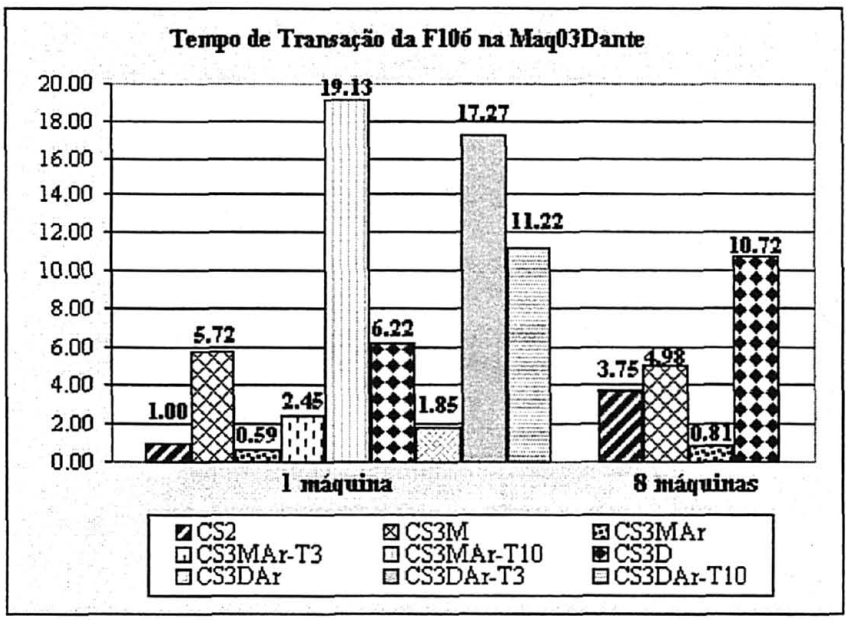


FIGURA 7.4 TEMPO DE TRANSAÇÃO PARA MAQ03DANTE

7.5.4 *Teste de segurança*

No teste de segurança foi projetado um exemplo simplificado da arquitetura e validados alguns *logins* para algumas situações. Como pode ser visto pelos resultados apresentados no APÊNDICE G, os resultados obtidos indicavam que faltava verificar detalhes na configuração dos pacotes do MTS. O teste permitiu verificar o problema, o qual depois de algumas tentativas foi descoberta a solução. E as observações quanto a estes detalhes ficaram documentadas juntamente com os resultados obtidos.

7.6 DOCUMENTAÇÃO DOS CASOS E RESULTADOS OBTIDOS

Os casos de testes executados foram documentados no Word e o arquivo gravado juntamente com os arquivos do projeto em um diretório chamado Testes. A maioria dos resultados obtidos estão registrados no APÊNDICE G e é importante documentar também os detalhes de configuração de ambiente que podem causar problemas na execução dos componentes, como foi o caso dos resultados obtidos através do teste de segurança, que foram identificados detalhes na criação e configuração dos pacotes e componentes no MTS apresentados no apêndice acima citado.

7.7 CONSIDERAÇÕES FINAIS

A ETACS foi aplicada em um estudo de caso para que a estratégia pudesse ser avaliada, verificando-se problemas no sistema com a aplicação dos testes, uma vez que este deve ser o objetivo principal do teste. E também para avaliar a estratégia neste ambiente e pudesse identificar refinamentos.

1. Para o teste de desempenho e estresse, foi verificado um erro "*3704 – Operation is not allowed when the object is closed*" para o qual não foi identificada a origem até o momento.
2. Problema encontrado quando executando o teste de desempenho também foi que para executar o componente era razoável o tempo, mas para mostrá-lo demorava muito. Com isso surgiu a necessidade de fazer mais alguns testes com o programa alterado e constatou-se que dependendo da estrutura de retorno, o MTS e o cliente fazem muitas trocas de informações dependendo do tipo de estrutura utilizado. Isso só foi possível identificar com o teste de desempenho.
3. Testes são extremamente importantes para definir a arquitetura no ambiente cliente-servidor, principalmente os testes de sistema (desempenho e segurança). Pode ser verificado pela aplicação da estratégia que o desempenho é muito influenciado pela arquitetura e o desempenho é um dos requisitos não funcionais que avaliam a qualidade do software.
4. Quando arquiteturas cliente-servidor 3 camadas estiverem sendo testadas para avaliar desempenho, deve-se avaliá-las sob estresse e em intervalos de tempo, porque esta característica influencia muito. Este item foi um refinamento da estratégia que não estava previsto e com a aplicação dela descobriu-se essa necessidade.

8 CONCLUSÕES E TRABALHOS FUTUROS

A atividade de teste é muito importante para melhora da qualidade dos sistemas produzidos ou gerenciados por qualquer empresa. Teste é necessário para atingir os níveis mais altos de qualidade, e o grau de certificação que toda empresa almeja. Toda empresa deseja que seus produtos sejam formalmente testados com critério. Um teste feito ao acaso estará fadado a um desastre quase inevitável. O planejamento não é a garantia de que tudo correrá tranqüilamente, mas é um passo necessário que aumenta muito as chances do sucesso.

A natureza distribuída do ambiente cliente servidor com toda a sua complexidade tornam mais difícil testá-lo e representa um grande desafio. A existência de uma estratégia de testes para estes casos representa um avanço na área de teste de software.

A maior dificuldade no ambiente cliente-servidor é o número de produtos envolvidos. E muitas vezes por não saber como começar nem se faz. A ETACS é mais que um Plano de Teste. Procura orientar a tarefa de teste desde o início do desenvolvimento do projeto. A estratégia proposta é independente de ferramentas mas, pode ser aplicada com auxílio delas. No entanto, procura ser prática e útil, organizando a tarefa para resolver os principais problemas. Tem por objetivo ser completa para orientar os vários passos da atividade de teste ao longo do desenvolvimento do projeto com roteiros, até chegar na aplicação dos critérios e técnicas existentes.

Conforme a necessidade apontada pela pesquisa de campo e resultados coletados da aplicação da estratégia, foram identificados problemas no teste de desempenho e nos testes de unidade também. Um dos problemas foi a dificuldade de simular o ambiente com volume de dados e número de usuários.

A ETACS mostrou através da aplicação em um estudo de caso que esta simulação é possível e deve ser feita, mesmo sem o auxílio de uma ferramenta. E que isto é bastante importante para resolver os problemas apontados e definir uma arquitetura adequada para as características do software que vai ser produzido. A definição correta desta arquitetura é

fator essencial e pode ser feita através dos testes de sistema. Na arquitetura cliente-servidor 3 camadas são os tipos de testes que causam maiores impactos na qualidade final do produto.

Os testes tradicionais apontados na literatura não são descartados; eles podem ser totalmente utilizados neste tipo de arquitetura, considerando uma unidade um componente, uma classe, uma função. Este foi item de maior dificuldade, pois uma das metas era definir uma tabela base para seleção dos critérios com base na prioridade. Este item foi muito difícil de aplicar, pois a seleção do critério depende diretamente da lógica do código ou das características do software e mesmo todos sendo para ambiente cliente-servidor não foi possível generalizar.

Uma dificuldade é que é necessário conhecer vários critérios para que se possa selecionar o mais adequado para que a estratégia possa ser aplicada de maneira eficiente. É uma dificuldade que produz uma característica de flexibilidade à estratégia, pois não é rígido na utilização de um critério específico e sim possibilita que sejam usados os critérios mais adequados ou ainda outros que venham a surgir.

É importante a adoção de uma ferramenta para a execução da atividade de teste ou parte dela. Se não é possível adotar uma ferramenta para auxiliar toda a atividade, adota-se ferramentas para testes específicos como o teste de estresse por exemplo. Diminui o trabalho repetitivo, aumenta a qualidade do teste e o tempo pode ser melhor aproveitado pois os testes podem ser executados em horários fora de expediente. A estratégia não está vinculada a uma ferramenta específica.

Como trabalhos futuros tem-se:

1. transformar esta estratégia em um roteiro para a empresa;
2. divulgar e treinar os funcionários para que a ETACS seja aplicada em outros projetos dentro da empresa para adquirir *know-how* e possa ser disseminada fora dela também;
3. divulgação dos conceitos de teste e das técnicas;
4. aplicação da ETACS em novos projetos e novos refinamentos.

REFERÊNCIAS

[1] AGUIAR, M. Teste de software, melhoria de processos e body shop. **Developers' Cio Magazine**, Rio de Janeiro, v. 5, n. 49, p. 14-18, set. 2000.

[2] AMARO, A. Encarando o teste de sistemas como um projeto. **Developers' Cio Magazine**, Rio de Janeiro, v. 4, n. 37, p. 36 e 37, set. 1999.

[3] ATKINS, W.; SHAW, J. C. **Managing computer system projects.** McGraw Hill Book, 1980.

[4] BEIZER, B. **Software system testing and quality assurance.** New York: Van Nostrand Reinhold, 1984.

[5] BEIZER, B. **Software testing techniques.** 2. ed. New York: Van Nostrand Reinhold, 1990.

[6] BENDER, R. **SEI / CMM proposed software evaluation and test KPA:** Revision 4. Larkspur: Bender and Associates, Apr. 1996.

[7] BONIFÁCIO, J. M. As tecnologias por trás de uma aplicação web 3-camadas. **Developers' Cio Magazine**, Rio de Janeiro, v. 5, n. 52, p. 24 e 25, dez. 2000.

[8] BUDD, T. A. **Mutation analysis:** ideas, examples, problems and prospects, computer program testing. North-Holand Publishing, 1981.

[9] COMPUWARE CORPORATION. Disponível em: <http://www.compuware.com>. Acesso em: 20/07/2001.

[10] CONWAY, B. **Testing client/server applications:** challenges, strategies and solutions for success. Stamford: Gartner Group, 22 Nov. 1996. (Strategic Analysis Report).

[11] CYRANO COMPANY. Disponível em: <http://www.cyrano.com/products>. Acesso em 20/07/2001.

[12] DEMILLO, R. A; LIPTON, R. J. Hints on test data selection: help for practicing programmer. **IEEE Computer**, v. 11, n. 4, p. 34-41, 1978.

[13] EDELSTEIN, H. Second-generation client/server. **DBMS Database & client/server solutions**, v. 8, n. 9, p. 60-68, 1995.

[14] FRANKL, P. G.; WEYUKER, E. J. An applicable family of dataflow criteria. **IEEE Transactions on software engineering**, p. 483-498, Oct. 1988.

[15] GRAHAM, D. R. Testing. In: ENCYCLOPEDIA of software engineering. J. Wiley, 1994. v. 2, p. 1330-1353.

[16] HALEY, A.; ZWEBEN, S. Development and aplication of a white box approach to integration testing. **The journal of systems and software**, n. 4, 1984.

[17] HARROLD, M. J.; SOFFA, M. L. Selecting and using data for integration testing. **IEEE Software**, v. 8, n. 2, p. 58-65, mar. 1991.

[18] HAYES, I. S.; ULRICH, W. M. **A crise do software no ano 2000 o desafio do século.** São Paulo: Makron Books, 1998.

[19] HERSHOVITZ, C. E. Mission-critical testing involving customers in a plan to deliver quality applications. **Component Strategies**, New York, v. 1, n. 12, p. 18-23, June 1999.

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

[21] HOWDEN, W. E. Methodology for the generation of program test data. **IEEE Transactions on software engineering**, v. 24, n. 5, p. 554-559, May 1975.

[22] HUMPHREY, W. S. **Managing the software process.** Massachusetts: Addison-Wesley, 1989.

[23] IEEE Glossary of software engineering terminology– ANSI/IEEE Std 729-1993. New York: CS Press, 1997. (IEEE Standard Collection Software Engineering).

[24] IEEE Guide for software verification and validation plans – ANSI/IEEE Std 1059-1993. New York: CS Press, 1997. (IEEE Standard Collection Software Engineering).

[25] IEEE Standard for software test documentation – ANSI/IEEE Std 829-1983. New York: CS Press, 1997. (IEEE Standard Collection Software Engineering).

[26] IEEE Standard for software unit testing – ANSI/IEEE Std 1008-1987. New York: CS Press, 1997. (IEEE Standard Collection Software Engineering).

[27] IEEE Standard for software verification and validation plans – ANSI/IEEE Std 1012-1986. New York: CS Press, 1997. (IEEE Standard Collection Software Engineering).

[28] IEEE Standard Glossary of Software Engineering Terminology. **Padrão 610.12.** New York: IEEE CS Press, 1990.

[29] JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The unified software development process reading.** Massachusetts: Addison-Wesley, 1999. 463 p.

[30] KARLSSON, J. Software requirements prioritizing. In: INTERNATIONAL CONFERENCE ON REQUIREMENTS ENGINEERING, 2., 1996, Colorado. **Proceedings**... Los Alamitos: IEEE CSP, 1996. 6 p.

[31] KNOWLES, R. **Automatic testing systems and application.** McGraw Hill, 1976.

[32] KRUCHTEN, P. **A Rational development process:** white paper. Disponível em: <http://www.rational.com/products/rup/prodinfo/whitepapers/>.

[33] LINNENKUGEL, U.; MÜLLERBURG, M. Test data selection criteria for (software) integration testing In: INTERNATIONAL CONFERENCE ON SYSTEMS INTEGRATION, 1., 1999, Morristown. **Proceedings**... Morristown, Apr. 1990. p. 709-717.

[34] MACHADO, C. F.; REINEHR, S. S.; CALSAVARA, A.; BURNETT, R. C. Aderência do RUP à norma NBR ISO/IEC 12207. In: SIMPÓSIO INTERNACIONAL DE MELHORIA DE PROCESSO, 2., 2000, São Paulo. **Anais...** São Paulo: SIMPROS, 2000.

[35] MACIEL, T. M. M.; CAMPÊLO, G. M. C. Suporte de ferramentas CASE na implantação do CMM. **Developers' Cio Magazine**, Rio de Janeiro, v. 5, n. 48, p. 34-37, ago. 2000.

[36] MALDONADO, J. C. **Critérios potenciais usos:** uma contribuição ao teste estrutural de software. 1991. Tese (Doutorado) - DCA/FEE/UNICAMP, Campinas, jul. 1991.

[37] MALDONADO, J. C.; CHAIN, M. L.; JINO, M. Arquitetura de uma ferramenta de teste de software de apoio aos critérios potenciais usos. In: CONGRESSO NACIONAL DE INFORMÁTICA, 22., 1989, São Paulo. **Anais...** São Paulo: SUCESU, 1989.

[38] MALDONADO, J. C.; VINCENZI, A. M. R.; BARBOSA, E. F.; SOUZA, S. R. S.

Aspectos teóricos e empíricos de teste de cobertura de software. In: ESCOLA DE INFORMÁTICA DA SBC DA REGIÃO SUL, 6., maio 1998. **Anais**... Blumenau: SBC, 1998.

[39] MARTINS, V. **Descrição da arquitetura de sistemas de informação distribuída baseados em componentes.** Curitiba: PUC/PR, 2001. (Dissertação em elaboração).

[40] MARTINS, V. O processo unificado no desenvolvimento de software. **Bate-Byte**, Curitiba, n. 89, ago. 1999.

[41] MATHEWS, D. Q. **The design of the management information system.** 2. ed. New York: Petrocelli Books, 1974.

[42] MCCABE, T. A software complexity measure. **IEEE Transactions on Software Engineering**, v. 2, n. 6, p. 308-320, dez. 1976.

[43] MERCURY INTERACTIVE. Disponível em: < www.merc-int.com>. Acesso em: 20/07/2001.

[44] MICROSOFT. **The Windows interface guidelines for software design.** Washington: Microsoft Press, 1995.

[45] MOSLEY, D. J. **Client server software testing on the desktop and the web.** Indianapolis: Prentice Hall, 2000.

[46] MYERS, G. **The art of software testing.** New York: J. Wiley, 1979.

[47] NTAFOS, S.C. On required element testing. **IEEE Transactions on Software Engineering**, v. 10, n. 6, nov. 1984.

[48] ORFALI, R.; HARKEY, D.; EDWARDS, J. **The essential client/server survival guide.** 2. ed. New York: J. Wiley, 1996.

[49] PATTISON, T. **Programming distributed applications with COM and Microsoft Visual Basic 6.0.** 2. ed. Microsoft Press, 2000.

[50] PAULK, M. C.; WEBER, C. V.; CURTIS, B.; CHRISSIS, M. B. **The capability maturity model:** guidelines for improving the software process / CMU /SEI. Reading: Addison-Wesley, 1995.

[51] PIRES, A.; MOLINARI, L. O gerenciamento de software e o produto imprevisível. **Developers' Cio Magazine**, Rio de Janeiro, v. 5, n. 51, p. 32-36, nov. 2000.

[52] PRESSMAN, R. S. **Engenharia de software**. 4. ed. New York: McGraw-Hill, 1997.

[53] PRESSMAN, R. S. **Engenharia de software**. São Paulo: Makron Books, 1997.

[54] PRESSMAN, R. S. **Software engineering:** a practitioner's approach. New York: McGraw-Hill, 1992.

[55] RAPPS, S.; WEYUKER, E. J. Data flow analysis techniques for test data selection. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, Tókio, 1982. **Proceedings**... Tókio, 1982. p. 272-278.

[56] RAPPS, S.; WEYUKER, E. J. Selecting software test data using data flow information. **IEEE transactions on software engineering**, v.11, n. 4, abr. 1985.

[57] RATIONAL SOFTWARE. Disponível em: < www.rational.com>. Acesso em: 20/07/2001.

[58] ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de software** – teoria e prática. São Paulo: Prentice Hall, 2001.

[59] ROYCE, W. **Software project management.** 4. ed. USA: Addison-Wesley, 1999.

[60] SEGUE SOFTWARE INC. Disponível em: <www.segue.com>. Acesso em: 20/07/2001.

[61] SIMÕES, C. A. Planejamento, especificação e execução dos testes. **Developers' Cio Magazine**, Rio de Janeiro, v. 4, n. 40, p. 22-24, dez. 1999.

[62] SOFTBRIDGE INC. Disponível em: <http://www.softbridge.com>. Acesso em: 20/07/2001.

[63] SOFTWARE RESEARCH INC. Disponível em: <www.soft.com>. Acesso em: 20/07/2001.

[64] STATISTICA for Windows, vr 5.1 da StatSoft Inc, 1984-1996. Disponível em: <www.statsoftinc.com>. Acesso em: 20/6/2000.

[65] TEIXEIRA FILHO, J. **Gerência de projetos com novas tecnologias.** INSGHT INFORMAL [028 - 13/10/1999]. Disponível em: <http://www.informal.com.br/insight/insight28.html>. Acesso em: 20/11/2000.

[66] VALLE, A.; MARCINIUK, M.; MELHORETTO, S. M.; BURNETT, R. Um road map para métricas de software: definições e histórico. **Developers' Cio Magazine**, Rio de Janeiro, v. 5, n. 49, p. 28-32, set. 2000.

[67] VAZ, R. Rumo ao nível II da capability maturity model – CMM. **Developers' Cio Magazine**, Rio de Janeiro, v. 5, n. 49, p. 20-23, set. 2000.

[68] VERGILIO, S. R.; MALDONADO, J. C.; JINO, M. Caminhos não-executáveis na automação das atividades de teste. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 6., 1992, Gramado. **Anais**... Gramado, nov. 1992. p. 343-356.

[69] VILELA, P. R. S. **Critérios potenciais usos de integração definição e análise.** Campinas: DCA/FEEC/UNICAMP, abr. 1998.

[70] VINCENZI, A. M. R.; BARBOSA, E. F.; DELAMARO, M. E.; SOUZA, S. R. S.; MALDONADO, J. C. Critério análise de mutantes: estado atual e perspectivas. In: WORKSHOP DE PROJETO VALIDAÇÃO E TESTE DE SISTEMAS DE OPERAÇÃO. **Anais**... jan. 1997

[71] VYSSOTSKY, V. A. **Commom sense in designing testable software:** program test methods. Prentice-Hall, 1973. cap. 6, p. 41-48.

[72] WANGENHEIM, C. G.; WANGENHEIM, A. Mensuração no melhoramento da qualidade de software. **Developers' Cio Magazine**, Rio de Janeiro, v. 5, n. 52, p. 28-32, dez. 2000.

[73] WOODWARD, M.; HEDDLEY, D.; HENNEL, M. **Experience with path analysis and testing of programs.** In: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, v. 6, p. 278-286, May 1980.

[74] ZANLORENCI, E. P. **Descrição e qualificação de requisitos:** um modelo aplicável à análise e validação da informação. Dissertação (Mestrado) – PUC/PR, Curitiba, jul. 1999.

APÊNDICE A FERRAMENTAS DISPONÍVEIS COMERCIALMENTE

A Tabela A.1 apresenta algumas ferramentas disponíveis no mercado e que fornecem algum auxílio para ambiente cliente-servidor.

TABELA A.1 EXEMPLOS DE FERRAMENTAS DE TESTE DISPONÍVEIS COMERCIALMENTE

Empresa	Produtos	Ambientes de execução
Mercury Interactive [43]	LoadRunner® 6 – Teste de carga stress para empresas / WinRunner® 6 - Teste de regressão e funcional / XRunner® 6 - Os padrões no teste de *Interface GUI* / TestDirector® 6 - Gerenciamento de teste integrado e trajetória dos defeitos / TestSuite® - Solução integrada para testes em aplicações windows client/server / TestBytes™ - Gerador de dados de teste / Astra® LoadTest™ - Teste de carga para *Web* / Astra® QuickTest™ - Ferramenta de teste funcional para negócio eletrônico / Astra® SiteManager™ - Teste de regressão e ferramenta de gerenciamento de site *Web* / Astra SiteTest™ - Teste de carga e stress para *Web* / Topaz™ - Gerenciamento de desempenho para aplicações *Web*	*Visual Basic*, PowerBuilder, *Sql Windows*, SAP/R3, *Oracle* Developer 2000, *Borland - Delphi*, aplicações baseadas em *Web* (WebTest), *Java*, Natstar, ILOG Views, Magic, e todas as aplicações *Mainframe* e AS/400 via um programa de emulação de terminal tal como Wall Data's RUMBA™, Attachmate™, Irma™, etc. Usa uma linguagem de *Script* de teste como 'C-like'
Segue Software [60]	SilkTest™ - Teste de regressão e funcional / SilkPerformer™ - Teste de carga e stress/desempenho / SilkRadar ™ - Gerenciamento de teste e trajetória do defeito automático / LiveQuality™ - Soluções de Negócios eletrônicos / SilkControl™ - Gerenciamento de site *Web* e monitoramento e relatório de servidores de base de dados / SilkPilot - teste de unidade de objetos CORBA / SilkObserver - gerenciamento e monitoramento de transações ponto a ponto para aplicações CORBA / SilkMeter - medição de uso e controle de acesso / SilkRealizer - modelagem de sistema e cenário de teste / SilkDeveloper™ - SilkScope™, e SilkSpec™ - Desenvolvimento	*Visual Basic*, PowerBuilder, *Sql Windows*, aplicações *Web*, *Java*, também aplicações *Mainframe* e AS/400 *Applications* via emulação de terminal. Usa sua própria linguagem de *Script* - "4GL" – que é muito parecida com C++.
Rational [57]	SQA Robot™ - Teste de regressão e funcional / SQA LoadTest™ - Teste de carga e stress / SQA Manager™ - Gerenciamento de teste e trajetória do defeito / SQA Process™ - Metodologia de teste / pre-Vue-CS - Teste de desempenho e carga/stress / SQA SiteCheck™ - Teste de stress, regressão e funcional para *Web* / *Performance* Studio, Visual Test, Purify e Quantify	Apóia o teste de objetos e componentes Windows 32- e 16-bit, incluindo OLE Controls (OCXs), *Internet* ActiveX Controls, controles *Visual Basic* (VBXs), objetos *Visual Basic*, objetos PowerBuilder, controles Win32, etc. Tem sua própria linguagem de *Script*.
CompWare [9]	QARun™ (*GUI* Applications) - Teste de regressão e funcional / QALoad™ - Teste de carga de servidor *Web* / aplicação e Base de dados / QAStress™ - Teste de stress / QADirector™ & TrackRecord™ - Gerenciamento do teste e trajetória do defeito / QAHiperstation™ & QAPlayback™ - Teste de aplicações de legado	Auxilia o teste para aplicações *internet*, client/server, e aplicações tanto diretamente no *Mainframe* CICS & VTAM Quanto em PC WorkStation (QAHiperstation+™ & QAPlayback+™).
Software Research [63]	CAPBAK/MSW e SMARTS/MSW - TestWorks para Regressão / TCAT C/C++ , TCAT para *Java*/Windows - TestWorks para cobertura	MS/Windows (Win3.1), MS/Windows NT or MS/Windows '95/'98 (Win32) teste de regressão completo para *Web* Site e a maioria das plataformas *UNIX*.

SoftBridge [62]	TestAdvantage's Automation Engine, também conhecido como Automated Test Facility (ATF), é uma ferramenta de teste produzida pela SoftBridge e foi projetada para teste de aplicações de missão crítica. Suporta vários níveis e tipos de teste: teste de sistema, *GUI*, funcional, regressão, estresse, carga, desempenho, conflito, concorrência, bloqueio, integração de vários PC, único PC, work-flow.	É possível sua utilização com as linguagens: PowerBuilder (6.0, 5.0 and 4.0), *Visual Basic* (6.0, 5.0, 4.0 (16 and 32bit) and 3.0), VisualAge para Smalltalk (v. 4.0, 4.5) 3270, (Attachmate Extra!, RUMBA v 5.2 U00), 5250, (Attachmate RALLY!, RUMBA AS/400), Utilitário FMB, Desktop, 3270 Add-on, Full-Screen Add-on, DOS Add-on, Win-OS/2 Add-on, Argo Add-on, e em breve Unimap.
Cyrano [11]	*Impact* - Ferramenta que captura e guarda transações do *Sql Server*. / *Insight* - Analisador de chamadas clientes em base de dados / *Manager* - Planeja, gerencia e analisa todos os aspectos dos seus projetos de teste cliente-servidor / *Production* - Analisador de desempenho de base de dados que fornece uma análise completa das transações do *Sql Server* / *Robot* - Para teste de regressão e funcional de aplicações Windows. / *Standards* - Para especificar regras, convenções, e padrões de projeto. / *Test* - Teste de regressão funcional e teste de stress para alto volume permitindo o teste completo da organização incluindo rede como se fosse uma unidade. Para arquitetura três camadas. / *Timer* - Mede os tempos de respostas da precisão das aplicações vivas. / *Watcher* - Ferramenta de análise de base de dados para analisar o desempenho do sistema. / *WebTester* - Cria, mantém e executa teste de regressão e teste funcional, teste de escalabilidade e carga, teste de confiabilidade e disponibilidade para suas aplicações baseadas em *Web* / *WinScope* - Otimizador do desenvolvimento de aplicações cliente-servidor. / *Workbench* - analisa *queries* e transações executadas entre clientes e *Sql Server*s para otimizar. Tem outras.	Windows NT™, Windows® 95, e Windows® 3.x

APÊNDICE B QUESTIONÁRIO COM OBJETIVOS E RESPOSTAS

O questionário foi elaborado em 5 seções: Perfil do Testador, Importância do Teste, Execução dos Testes, Verificando Critérios e Técnicas e Identificando maiores dificuldades. As respostas apresentadas neste apêndice foram tabuladas independente do perfil. Por questões de preservação da identidade da empresa foram tiradas duas questões.

(A - PERFIL DO TESTADOR)

1) Qual sua principal área de atuação? (Se você se enquadra em mais de um perfil, responda mais de uma vez o questionário para cada perfil.)

(8 -> 16,7%) *O gerência/coordenação*
(34 -> 70,8%) *O análise de sistemas/líder de projeto*
(6 -> 12,5%) *O programação de sistemas*

***Objetivo**: Identificar que tipo de usuário está respondendo a questão para dar um peso, líder de projeto tem peso 3, programador tem peso 2, coordenação ou gerência tem peso 1*

2) Qual o seu maior objetivo quando você testa um sistema:

(22->45,8%) *O encontrar erros no seu programa*
(21->43,8%) *O comprovar que seu programa está correto*
(0 -> 0%) *O preencher tarefas do seu cronograma, cumprir etapas do planejamento*
(5 -> 10,4%) *O Outro: _____*

- *Verificar se o sistema está atendendo plenamente à necessidade*
- *Verificar se o sistema atende todas as regras especificadas*
- *Na posição de suporte, eu testo o que está acontecendo de errado conforme solicitação do cliente. Depois de analisar o programa e realizar possíveis modificações, voltam os testes em cima do problema relacionado para comprovar as melhorias.*
- *Com certeza existem vários objetivos no teste de sistema, entre eles encontrar erros. Mas o principal objetivo é assegurar que o sistema possua o menor nível de erros.*
- *além de verificar se o programa está correto, a sua funcionalidade.*

***Objetivo**: Verificar qual a motivação do testador quando realizando a sua tarefa*

3) Quantas horas do seu tempo são em média, gastas semanalmente com falhas de sistemas (correções de código, alterações de *layout* , atendimento a usuários, recuperando dados alterados incorretamente, com manutenção ou com retrabalho) em sistemas que já foram implantados, estando em produção.

(23 -> 47,9%) O menos de 4 horas *(13 -> 27,1%) O de 4 a 8 horas*
(8 -> 16,7%) O de 8 a 16 horas *(4 -> 8,3%) O mais de 16 horas*

***Objetivo**: Verificar quanto tempo é gasto com manutenção para calcular o custo do teste nesta fase e provar o quanto seria mais barato se o sistema não precisasse de tanta manutenção*

4) Quantas horas do seu tempo são em média, gastas semanalmente testando os programas antes de liberá-lo para o cliente (fase de implantação)

(10 -> 20,8%) O menos de 4 horas *(16 -> 33,3%) O de 4 a 8 horas*

(14 -> 29,2%) O de 8 a 16 horas *(7 -> 14,6 %) O mais de 16 horas*

(1 -> 2,1%) Branco

***Objetivo**: Fazer uma relação de quanto maior tempo gastar com teste antes da implantação menos se gasta com manutenção*

(B - IMPORTÂNCIA DO TESTE)

1) Em sua opinião, quanto a atividade de teste de um sistema...

(13 -> 27,1%) O devem ser testados os erros mais comuns e os mais críticos, que o usuário poderia cometer

(0 -> 0%) O não precisa ser executada pois os erros aparecem com a utilização do software

(5 -> 10,4%) O quando houver tempo devem ser executados testes mais criteriosos

(29 -> 60,4%) O necessariamente devem ser executados testes segundo uma metodologia e com muito critério

(1 -> 2,1%) Branco

***Objetivo**: Identificar qual a importância que o usuário dá para teste e fazer com que o usuário se comprometa*

2) Classifique as sentenças abaixo com V = Verdadeiro, F = Falso e D=Depende. E as respostas podem ser observadas através da tabela B.1.

TABELA B.2 RESPOSTAS DA QUESTAO 2 DA SEÇÃO B DO QUESTIONÁRIO

V	V%	F	F%	D	D%	B	B%	Item
2	4,2	41	85,4	5	10,4	0	0,0	- O teste deve ser sacrificado em decorrência do cronograma estar atrasado
26	54,2	8	16,7	14	29,2	0	0,0	- O programa deve ser desenvolvido exatamente conforme a lista de requisitos no RPRE
46	95,8	0	0,0	1	2,1	1	2,1	- O programa deve ser bem documentado e fácil de usar
35	72,9	10	20,8	3	6,3	0	0,0	- O teste deve ser planejado desde a fase de análise
40	83,3	4	8,3	4	8,3	0	0,0	- Deve existir uma fase especifica para se fazer testes logo após a programação ser concluida
38	79,2	2	4,2	7	14,6	1	2,1	- A imagem da empresa é comprometida pois se o programa tem erros ela não é confiável
3	6,3	39	81,3	6	12,5	0	0,0	- É normal ocorrerem erros e isso não compromete a imagem da empresa pois os clientes entendem isso
3	6,3	33	68,8	11	22,9	1	2,1	- Os usuários não chegam a ficar insatisfeitos quando encontram erros porque você os conserta
0	0,0	47	97,9	1	2,1	0	0,0	- Teste está ligado apenas às atividades de verificação de código do programa
35	72,9	7	14,6	6	12,5	0	0,0	- Deve ser incorporado a todas as fases do projeto, como um procedimento de revisão
0	0,0	48	100,0	0	0,0	0	0,0	- Teste é um exagero ligado à qualidade de software
4	8,3	38	79,2	5	10,4	1	2,1	- É mais barato testar o projeto em uma fase depois da programaçcão pois teste custa caro e o usuário final pode realizar

***Objetivo**: Verificar se o usuário tem consciência da importância do teste e principalmente relembrá-lo de conceitos e conseqüências de não testar adequadamente que são importantes mas muitas vezes esquecidos*

(C - EXECUÇÃO DO TESTE)

1) Você desenvolve um plano estabelecendo objetivos e respondendo às perguntas "o que testar" e "quando terminar"?

(7 -> 14,6%) O Sempre (16 -> 33,3%) O Quase sempre (22 -> 45,8%) O Às vezes
(3 -> 6,3%) O Nunca (0 –> 0% em branco)

***Objetivo**: Comprovar que muito poucos planejam o teste*

2) Em que etapa do desenvolvimento você começa a se preocupar em como você vai testar o sistema, desenvolvendo um plano de teste?

(2 -> 4,2%) O Análise/Especificação Requisitos
(3 -> 6,3%) O Análise/Definição da solução do sistema
(11 -> 22,9%) O Desenvolvimento (começo – inicia os testes das funções/partes que vão ficando prontas)
(2 -> 4,2%) O Desenvolvimento (terminado – inicia fase de teste antes da implantação)
(30 -> 62,5%) O Branco

***Objetivo**: Comprovar se começa a pensar em teste muito tarde*

3) Você mantém uma lista do que é efetivamente testado e dos resultados obtidos?

(6 -> 12,5%) O Sempre (14 -> 29,2%) O Quase sempre (19 -> 39,6%) O Às vezes
(7 -> 14,6%) O Nunca (2 –> 4,2% em branco)

***Objetivo**: Verificar se é feito um histórico de teste para se fazer teste de regressão*

4) Como essa lista é armazenada? (Se a resposta da questão anterior foi diferente de nunca)

(1 -> 2,1%) O Notes (0 -> 0%) O Doc (Word, Excel, etc) (3 -> 6,3%) O Rascunho
(11 -> 22,9%) O Outros: __ (33 -> 68,8% em branco)

***Objetivo**: Verificar onde se é arquivado os testes ou se os dados são perdidos após a implantação do sistema não possibilitando fazer teste de regressão*

5) Você mantém um registro do tempo gasto e dos recursos utilizados durante o teste (antes de estar em produção), servindo para calcular o custo total do teste? Onde você registra? _____

(5 -> 11,1%) O Sempre (4 -> 8,9%) O Quase sempre (14 -> 31,1%) O Às vezes
(22 -> 48,9%) O Nunca (0 –> 0% em branco)

***Objetivo**: Verificar onde são guardados os custos do teste*

6) Você mantém um registro dos erros encontrados durante o teste, servindo para avaliar os benefícios resultantes da realização do teste?

(6 -> 12,5%) O Sempre (3 -> 6,3%) O Quase sempre (18 -> 37,5%) O Às vezes
(21 -> 43,8%) O Nunca (0 –> 0% em branco)

***Objetivo**: Verificar se o testador tem documentado quão eficiente/importante foi seu teste*

7) Você conduz os testes como uma atividade sistemática e organizada, segundo uma metodologia bem definida e planejada?

(5 -> 10,4%) O Sempre (8 -> 16,7%) O Quase sempre (22 -> 45,8%) O Às vezes
(13 -> 27,1%) O Nunca (0 –> 0% em branco)

***Objetivo**: Identificar se tem pessoas que realizam testes criteriosos ou se é feito um teste ad-hoc*

8) Descreva com o máximo de detalhes possíveis como você realiza os testes: _____

Objetivo: *Verificar como as pessoas realizam os testes*

9) Você faz revisões periódicas dos planos e registros de teste?

(2 -> 4,2%) O Sempre (9 -> 18,8%) O Quase sempre (18 -> 37,5%) O Às vezes
(18 -> 37,5%) O Nunca (1 –> 2,1% em branco)

Objetivo: *Comprovar que quando pessoas fazem planos de teste são mais para preencher etapas do que para realizar bom casos de teste*

10) Quando você faz alguma manutenção no programa você refaz todos os testes feitos anteriormente para garantir que a alteração não provocou nenhum impacto em outro lugar do sistema?

(10 -> 20,8%) O Sempre (12 -> 25,0%) O Quase sempre (18 -> 37,5%) O Às vezes
(8 -> 16,7%) O Nunca (0 –> 0% em branco)

Objetivo: *Verificar se é feito teste de regressão*

11) Você gera dados de teste com base na especificação de requisitos obtidas através do cliente?

(13 -> 27,1%) O Sempre (21 -> 43,8%) O Quase sempre (12 -> 25,0%) O Às vezes
(2 -> 4,2%) O Nunca (0 –> 0% em branco)

Objetivo: *Verificar se os dados de teste são originados dos requisitos dos clientes*

12) Você gera dados de teste com base no código fonte do programa?

(8 -> 16,7%) O Sempre (9 -> 18,8%) O Quase sempre (17 -> 35,4%) O Às vezes
(12 -> 25,0%) O Nunca (2 –> 4,2% em branco)

Objetivo: *Verificar se os dados de teste são originados da estrutura do programa*

13) Você testa o sistema inteiro, com entradas obtida a partir de sua experiência do que o usuário poderia digitar errado?

(16 -> 33,3%) O Sempre (15 -> 31,3%) O Quase sempre (15 -> 31,3%) O Às vezes
(1 -> 2,1%) O Nunca (1 -> 2,1% em branco)

Objetivo: *Verificar se os dados de teste são originados ad-hoc, sem critério*

(D - VERIFICAR CRITÉRIOS E TÉCNICAS)

1) Você testa alguma função específica através de um depurador (debugador) verificando conteúdo de variáveis?

(2 -> 4,2%) O Sempre (10 -> 20,8%) O Quase sempre (27 -> 56,3%) O Às vezes
(9 -> 18,8%) O Nunca (0 –> 0% em branco)

Objetivo: *Verificar é feito teste de unidade*

2) Faz uns testes iniciais e envia o sistema para o cliente para uma fase de teste?

(8 -> 16,7%) O Sempre (12 -> 25,0%) O Quase sempre (20 -> 41,7%) O Às vezes
(8 -> 16,7%) O Nunca (0 –> 0% em branco)

Objetivo: *Verificar é feito teste beta*

3) Você convida o cliente para fazer testes no seu ambiente de trabalho (ambiente de desenvolvimento)?

(3 -> 6,3%) O Sempre (10 -> 20,8%) O Quase sempre (18 -> 37,5%) O Às vezes
(17 -> 35,4%) O Nunca (0 –> 0% em branco)

***Objetivo**: Verificar é feito teste alfa*

4) Quando você testa no seu código, um conjunto de funções, você o faz testando isoladamente uma e depois integrando ao conjunto restante do sistema?

(19 -> 39,6%) O Sempre (20 -> 41,7%) O Quase sempre (8 -> 16,7%) O Às vezes
(0 -> 0%) O Nunca (1 –> 2,1% em branco)

***Objetivo**: Verificar é feito teste de integração*

5) Você testa se o sistema tem bom desempenho em situações de "pico" (muitos usuários, muitos dados)?

(2 -> 4,2%) O Sempre (6 -> 12,5%) O Quase sempre (26 -> 54,2%) O Às vezes
(14 -> 29,2%) O Nunca (0 –> 0% em branco)

***Objetivo**: Verificar é feito teste de* estresse/desempenho

(E - IDENTIFICANDO MAIORES DIFICULDADES)

1) Qual é o problema que você mais encontra nos sistemas em produção? Outros foi respondido junto com as opções por algumas pessoas e foi registrado aqui.

(27 -> 56,3%) O Erros de código (19 -> 39,6%) O Desempenho (6 -> 12,5%) O Travamento
(17 -> 35,4%) O Outro:_____

- *Incompatibilidade do ambiente windows*
- *problema de execução de procedures de produção – poucos*
- *Alterações sem analise de impactos, em alguma parte do sistema, que implica em outras partes do sistema ou outros sistemas; programa depende de dados em tabela, e tabela não foi atualizada (UFIR/Feriados); Registros em duplicidade; Duplicidade de processamento;*
- *Desconhecimento dos requisitos por parte do próprio cliente*
- *situação não prevista*
- *forte integração entre diversos sistemas sem o devido conhecimento de todos os responsáveis (efeitos colaterais)*
- *erros cometidos pelo usuário. Alterações de layout*
- *alterações nos requisitos - manutenção*
- *dinâmica do estado, mudando conceitos e características*
- *no sistema que estou testando no momento encontramos problemas de lock de banco de dados, precisando reformular o acesso ao banco*
- *sistema não estar totalmente preparado para o usuário, este, devido a mau uso/desconhecimento, acaba gerando erros*
- *Um dos maiores problemas que um sistema pode ter em relação a complexidade de qualquer coisa, e principalmente teste, é o fato dele ter um nível alto de integração com outros sistemas. A complexidade aumenta se esta integração se amplia para sistemas fora da sua equipe de desenvolvimento e até mesmo fora da empresa.*
- *alteração no escopo do projeto inicial e necessidade de adequação do sistema atual*
- *falhas na cooperação de sistemas batch, apesar da documentação estar atualizada. Registros com lixo (consistências não previstas)*
- *dados incorretos gerados pelo cliente. Processamento errado de procs pela produção. Mudanças desconhecidas nas gerações de informações nos arquivos base (mainframe), abendando o procedimento de coleta de informações*
- *integração - informações vindas de outros sistemas*
- *erros de códigos + falta de conhecimento no sistema + falha na especificação dos requisitos*

Objetivo: *Identificar onde está a maior dificuldade para tentar se concentrar nos tipos de teste mais eficientes para atacar este problema*

2) Que tipo de teste seria mais necessário para o seu trabalho? Coloque em ordem, (1) para o mais importante (2) para o segundo, e assim por diante até o menos importante. Essas respostas foi calculado a média para cada item de acordo com o preenchimento e o resultado apresentado a seguir:

2,0 c) Testar se o sistema em desenvolvimento realiza todos os requisitos do cliente
2,6 d) Encontrar o maior número possível de erros de lógica no seu sistema em desenvolvimento
3,9 e) Testar se o sistema está executando todas as camadas, cliente, middelware e bd corretamente
4,0 b) Testar se o sistema em produção vai suportar um número grande de informações/dados sem travar
4,2 f) Testar os aspectos de segurança, tentativas de burlar os caminhos naturais para se obter os dados, hackers
5,0 a) Testar se o sistema em produção vai suportar x número de usuários ao mesmo tempo
5,3 g) Testar se o sistema é consistente quando há alguma falha de energia, ou algo assim em uma das camadas
6,5 h) Outro tipo de teste:

- *funcionalidade/usabilidade/interface amigável*
- *interface sistema x usuário*
- *desempenho com grandes quantidades de dados*
- *teste de integração com outros sistemas*
- *facilidade de uso, funcionalidade, fácil de entender (todos os problemas relacionados com o sistema são de grande importância)*
- *integração*
- *verificar desempenho quando o volume de dados é grande para antecipar problemas futuros*

Objetivo: *Verificar que tipos de teste seriam mais eficientes ou importantes para resolver os maiores problemas encontrados*

3) Cite todas as ferramenta que poderiam auxiliar a atividade de teste que você conhece ou ouviu falar?_____

- *debug/trace das ferramentas*
- *debug*
- *Requisite Pro*
- *VB Debug, Clipper, cliente de BDs, DBU, Delphi Watchs*
- *Debug de ferramentas*
- *Desenvolveu uma ferramenta para fazer carga*
- *Debug de ferramentas*
- *tabela de decisão, português estruturado árvore e pseudocódigo*

Objetivo: *Identificar as ferramentas existentes para estudar no que elas podem ajudar*

4) Quais as maiores dificuldades que você tem para testar?

(20 -> 41,7%) ☐ falta de um roteiro

(36 -> 75,0%) ☐ você não lembra todos as entradas para verificar todas as situações possíveis

(8 -> 16,7%) ☐ você não planejou um teste com valores de entrada e resultados que deveriam ocorrer

(9 -> 18,8%) ☐ você não consegue simular o ambiente do cliente com número de usuários e carga de dados grande

*(9 -> 18,8%) ☐ Outros*_____

- *Fazer o cliente se envolver nos testes*
- *cronograma*
- *prazos de entrega ao cliente*

- *planejamento com entendimento do cliente, que o tempo para testes NUNCA deve ser abolido ou negociado*
- *conseguir preparar dados coerentes no ambiente de teste de forma a fazer uma simulação real possível*
- *prazo para entrega, dificulta um teste mais aprimorado*
- *falta de cultura voltada considerar o teste como uma atividade nobre - geralmente as pessoas vêem como uma atividade chata*
- *nem sempre tem um ambiente de teste, acaba sendo necessário testar em produção*

Objetivo: *Identificar as dificuldades para tentar propor estratégias para minimizá-las*

5) O que você imagina que possa ser feito para melhorar a atividade de teste

(36 -> 75,0%) ☐ ferramenta (17 -> 35,4%) ☐ metodologia (25 -> 52,1%) ☐ roteiro
(31 -> 64,6%) ☐ treinamento (9 -> 18,8%) ☐ Outros: (a seguir)

- *melhorar negociação dos prazos de entrega*
- *palestras de conscientização dos técnicos para o problema. Testes são problemas normalmente de imaturidade dos técnicos envolvidos.*
- *criar uma equipe similar a de Documentação, com usuários leigos para testar sistemas, criar e disseminar a cultura de que o período de testes é fundamental e deve ocorrer sempre; criar um ambiente de desenvolvimento de sistemas e, assim, conseguir efetuar testes em diversos locais e com usuários diferentes, sem a necessidade de....*
- *mapeamento das integrações do sistema com outros sistemas*
- *roteiro simplificado*
- *falta tudo em se tratando de testes... Cada um faz de um jeito diferente*
- *ter ambiente permanente e conhecido de teste e dados, agilizando o processo de teste*
- *a programação testar se o programa está passando por todas as rotinas especificadas. A análise poderá se preocupar mais com os aspectos de cada informação relacionada ao conjunto*
- *disponibilidade de recursos de hardware para ambiente de teste*

Objetivo: *Obter sugestões de melhoria que poderão ser usada nas estratégias que serão propostas*

6) Quais as linguagens de desenvolvimento que você tem usado atualmente (último ano)?

(17 -> 35,4%) ☐ Access (6 -> 12,5%) ☐ Clipper (7 ->14,6%) ☐ Cobol (2 -> 4,2%) ☐ C++
(19 -> 39,6%) ☐ Delphi (6 -> 12,5%) ☐ Java (30 -> 62,5%) ☐ Natural (13 -> 27,1%) ☐ VB
(7 -> 14,6%) ☐ Sql Windows (15 -> 31,3%) ☐ Notes Script (6 -> 12,5%) ☐ Outros

Objetivo: *Verificar quais as linguagens que estão sendo utilizadas pelos funcionários para verificar a possibilidade de adequar a estratégia a elas*

7) Você se considera nas linguagens que você assinalou no item anterior?

(4 -> 8,3%) O Iniciante (31 -> 64,6%) O Intermediário (13 -> 27,1%) O Expert

Objetivo: *Verificar o grau de maturidade em desenvolvimento os técnicos possuem*

8) Quanto tempo você tem trabalhado nas tecnologias que você assinalou?

(5 -> 10,4%) O até 1 ano (9 -> 18,8%) O de 1 a 2 anos (13 -> 27,1%) O de 3 a 5 anos
(21 -> 43,8%) O mais de 5 anos

Objetivo: *Verificar se o grau de maturidade que eles assinalaram corresponde a um tempo aceitável de desenvolvimento*

APÊNDICE C AVALIAÇÃO E JUSTIFICATIVA DOS ITENS

Quais os **efeitos** se acontecer alguma falha para cada caso de uso, o que acontece se algum caso de uso falhar.

TABELA C.1 AVALIAÇÃO E JUSTIFICATIVA DO ITEM 1 - EFEITOS

Caso de uso	Efeito	Grau do efeito	Justificativa
1. Cadastrar Distribuição das multas	Não será possível os órgãos consultarem o que devem e o que tem em haver	Regular	Isso pode ser feito manualmente como é atualmente feito, mas somente os valores totais sem discriminações e para isso existe uma relação de confiança
	Dados podem estar inconsistentes com a realidade	Crítico	Pode gerar uma diferença nos valores e não ser possível um fácil descobrimento do erro.
2. Excluir a Distribuição	Informação desnecessária vai aparecer, dados não válidos estarão ocupando espaço no banco	Regular	Nas consultas não vão aparecer pois não estarão na restrição, espaço no banco não é significativo pois as informações são relativamente curtas não tem imagem ou arquivo binário.
	For executado parcialmente vai gerar dados inconsistentes	Crítico	Em alguma consulta inclusive para repasse de dinheiro pode estar presente parte desta distribuição que não é mais direito.
3. Estornar Pagamento	Ficam dados inválidos no sistema e não devolve o dinheiro para usuário que pagou a mais	Prioritário	Pode ser feito manualmente mas é muito trabalhoso o serviço e o cliente fica aguardando receber dinheiro que tem em haver
4. Consultar Árvore de Distribuição por Docto	Desconforto para o usuário que não terá a informação quando precisa	Prioritário	Consulta pode ser feita junto aos órgãos, demora alguns dias para fazer o levantamento, por ser bastante complexa a consulta, mas é possível fazê-la manualmente
5. Consultar Distribuição Guia-Auto	Desconforto para o usuário que não terá a informação quando precisa	Pouca importância	Consulta pode ser feita junto aos órgãos, demora algumas horas para fazer o levantamento e é possível fazê-la manualmente
6. Consultar Distribuição por Auto	Desconforto para o usuário que não terá a informação quando precisa	Pouca importância	Consulta pode ser feita junto aos órgãos, demora algumas horas para fazer o levantamento mas é possível fazê-la manualmente
7. Gerar Perfil de Arrecadação	Desconforto para o usuário que não terá a informação quando precisa	Prioritário	Alguns tipos de consultas não podem funcionar
8. Auditar Pagamento sem Crédito	Aumenta o risco de irregularidades	Regular	Pode ser feita uma análise manual mas é mais fácil de camuflar alguma irregularidade

Quais as possíveis **causas** dos defeitos, o que pode provocar algum defeito, uma saída não desejada causada por uma falha neste caso de uso.

TABELA C.2 AVALIAÇÃO E JUSTIFICATIVA DO ITEM 2 - CAUSAS

Caso de uso	Causa	Grau do efeito	Justificativa
1. Cadastrar Distribuição das multas	Cadastrar duas vezes o mesmo documento	Crítico	Porque os dados serão inconsistentes, quando algum órgão for repassar o dinheiro vai repassar mais do que na realidade tem
	Cai conexão com o servidor de banco quando fazendo o cadastro	Regular	Programar para que ao atualizar várias tabelas seja feito em uma única transação, e não fiquem dados inconsistentes no banco
	Dados inválidos	Regular	Devem ser validados para não permitir uma tentativa de entrada com dados incorretos e uma verificação no banco que garanta que nada foi esquecido
2. Excluir a Distribuição	Conexão com banco perdida	Regular	Pode ser desenvolvida uma forma de garantir a transação pelo monitor de transação para auxiliar o servidor de dados
3. Estornar Pagamento	Conexão com banco perdida	Regular	Pode ser desenvolvida uma forma de garantir a transação pelo monitor de transação para auxiliar o servidor de dados
4. Consultar Árvore de Distribuição por Docto	Conexão com banco perdida Dados incorretos	Crítico	Pode ser desenvolvida uma forma de garantir a transação pelo monitor de transação para auxiliar o servidor de dados Validação dos dados e consulta muito pesada pode sobrecarregar o banco
5. Consultar Distribuição Guia-Auto	Conexão com banco perdida Dados incorretos	Prioritário	Pode ser desenvolvida uma forma de garantir a transação pelo monitor de transação para auxiliar o servidor de dados Validação dos dados
6. Consultar Distribuição por Auto	Conexão com banco perdida Dados incorretos	Prioritário	Pode ser desenvolvida uma forma de garantir a transação pelo monitor de transação para auxiliar o servidor de dados Validação dos dados
7. Gerar Perfil de Arrecadação	Conexão com banco perdida	Regular	Pode ser desenvolvida uma forma de garantir a transação pelo monitor de transação para auxiliar o servidor de dados
8. Auditar Pagamento sem Crédito	Conexão com banco perdida	Pouca importância	Não é algo que tenha que ser feito em um dado momento com urgência e informações podem ser obtidas por outras vias

Quais as **probabilidades** deste caso de uso falhar, de acontecer as causas que provocariam uma falha.

TABELA C.3 AVALIAÇÃO E JUSTIFICATIVA DO ITEM 3 - PROBABILIDADES

Caso de uso	Probabilidades	Grau do efeito	Justificativa
1. Cadastrar Distribuição das multas	Remota	Pouca importância	O arquivo *batch* vai ser gerado por *mainframe*, automatizado, não é nada normal os dados estarem inconsistentes no mainframe
2. Excluir a Distribuição	Remota	Regular	O servidor está normalmente no ar, e não é comum cair a conexão
3. Estornar Pagamento	Remota	Regular	O servidor está normalmente no ar, e não é comum cair a conexão
4. Consultar Árvore de Distribuição por Guia	Alta	Prioritário	Alta possibilidade de ocorrer alguma falha de conexão ou entrada de dados inválidos ou conexão com *internet* perdida
5. Consultar Distribuição Guia-Auto	Alta	Prioritário	Alta possibilidade de ocorrer alguma falha de conexão ou entrada de dados inválidos ou conexão com *internet* perdida
6. Consultar Distribuição por Auto	Alta	Prioritário	Alta possibilidade de ocorrer alguma falha de conexão ou entrada de dados inválidos ou conexão com *internet* perdida
7. Gerar Perfil de Arrecadação	Remota	Regular	O servidor está normalmente no ar, e não é comum cair a conexão
8. Auditar Pagamento sem Crédito	Remota	Exceção de qualidade	Como é feito esporadicamente pelo próprio sistema, só vai ocorrer quando estiver tudo normal.

Este caso de uso é utilizado por **quantos usuários**? Qual o **número de vezes** que este caso de uso é executado em um dado período de tempo (no mês)?

TABELA C.4 AVALIAÇÃO E JUSTIFICATIVA DO ITEM 4 - QTDE USUÁRIOS

Caso de uso	Número de Acessos	Grau do efeito	Justificativa
1. Cadastrar Distribuição das multas	Sistema em batch em torno de 10.000 transações por semana	Prioritário	É realizada por poucas pessoas mas com muita freqüência
2. Excluir a Distribuição	Sistema realiza em batch os acertos uns 100 casos por mês	Regular	O servidor está normalmente no ar
3. Estornar Pagamento	Sistema realiza em batch os acertos uns 20 casos por mês	Pouca importância	Poucas pessoas terão acesso a esta função que é realizada não com muita freqüência
4. Consultar Árvore de Distribuição por Docto	200 órgãos acessando grande volume de informação	Crítico	Muitos órgãos acessando e possibilidade de usuários inexperientes entrarem com informação incorreta é grande
5. Consultar Distribuição Docto-Auto	200 órgãos acessando grande volume de informação	Prioritário	Muitos órgãos acessando e possibilidade de usuários inexperientes entrarem com informação incorreta é grande
6. Consultar Distribuição por Auto	200 órgãos acessando grande volume de informação	Crítico	Muitos órgãos acessando e possibilidade de usuários inexperientes entrarem com informação incorreta é grande
7. Gerar Perfil de Arrecadação	10 órgãos tem permissão mas é feito esporadicamente	Regular	Vários órgãos tem direito e interesse nessa função

8.	Auditar Pagamento sem Crédito	Feito progmaticamente pelos sistema a cada mês	Exceção qualidade	Poucas pessoas terão acesso a esta função que é realizada esporadicamente

Levar em consideração o **perfil do usuário** que vai usar o sistema.

TABELA C.5 AVALIAÇÃO E JUSTIFICATIVA DO ITEM 5 - PERFIL DO USUÁRIO

Caso de uso	perfil do usuário	Grau do efeito	Justificativa
1. Cadastrar Distribuição das multas	Exigência = 5 Experiência = 10 Sistema gera informações	Pouca importância	Esta função vai ser acessada somente pelo sistema e que mas é um pouco exigente quanto a ocorrência de uma falha
2. Excluir a Distribuição	Exigência = 5 Experiência = 10 Sistema gera informações	Pouca importância	Esta função vai ser acessada somente pelo sistema e que mas é um pouco exigente quanto a ocorrência de uma falha
3. Estornar Pagamento	Exigência = 5 Experiência = 10 Sistema gera informações	Pouca importância	Esta função vai ser acessada somente pelo sistema e que mas é um pouco exigente quanto a ocorrência de uma falha
4. Consultar Árvore de Distribuição por Docto	Exigência = 8 Experiência = 3	Prioritário	Esta função é utilizada por vários órgãos e a possibilidade de algum usuário inexperiente entrar com dados incorretos é grande
5. Consultar Distribuição Docto-Auto	Exigência = 8 Experiência = 3	Prioritário	Esta função é utilizada por vários órgãos e a possibilidade de algum usuário inexperiente entrar com dados incorretos é grande
6. Consultar Distribuição por Auto	Exigência = 8 Experiência = 3	Prioritário	Esta função é utilizada por vários órgãos e a possibilidade de algum usuário inexperiente entrar com dados incorretos é grande
7. Gerar Perfil de Arrecadação	Exigência = 5 Experiência = 5	Regular	Esta função vai ser acessada somente por pessoas experientes e que não são tão exigentes quanto a ocorrência de uma falha
8. Auditar Pagamento sem Crédito	Exigência = 0 Experiência = 10	Exceção de qualidade	Esta função vai ser acessada somente pelo sistema e que não é exigente quanto a ocorrência de uma falha

Levar em consideração questões de **contrato** entre cliente e fornecedor. Verificar se o sistema pode ser entregue sem este caso de uso em questão.

TABELA C.6 AVALIAÇÃO E JUSTIFICATIVA DO ITEM 6 - CONTRATO

Caso de uso	contrato	Grau do efeito	Justificativa
1. Cadastrar Distribuição das multas	Essencial	Crítico	Sem esta função nada mais acontece no sistema
2. Excluir a Distribuição	Há meios manuais mas trabalhosos	Pouca importância	Esta função pode ser feita por um DBA e não é tão freqüente.
3. Estornar Pagamento	Há meios manuais mas trabalhosos	Pouca importância	Esta função pode ser feita por um DBA e não é tão freqüente.
4. Consultar Árvore de Distribuição por Docto	Muito importante	Prioritário	Importante mas podem ser entregues versões parciais sem ela
5. Consultar Distribuição Docto-Auto	Muito importante	Prioritário	Importante mas podem ser entregues versões parciais sem ela
6. Consultar Distribuição por Auto	Muito importante	Prioritário	Importante mas podem ser entregues versões parciais sem ela
7. Gerar Perfil de Arrecadação	Não muito importante	Regular	Importante mas podem ser entregues versões parciais sem ela
8. Auditar Pagamento sem Crédito	Pode ser realizada manualmente	Exceção de qualidade	Pode ser realizada manualmente

APÊNDICE D CÁLCULO DO GRAU DE PRIORIDADE

O embasamento para a aplicação destes cálculos está apresentado na Seção 6.2.2.

1. Foram copiados todos os conceitos identificados nos itens das Tabelas C.1, C.2, C.3, C.4, C.5 e C.6, conforme a tabela a seguir:

TABELA D.1 CONCEITOS IDENTIFICADOS PARA CADA CASO DE USO COM TODOS OS ITENS

Casos de uso	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
1. Cadastrar Distribuição das multas	Regular Crítico	Crítico Regular Regular	Pouca importância	Prioritário	Pouca importância	Crítico
2. Excluir a Distribuição	Regular Crítico	Regular	Regular	Regular	Pouca importância	Pouca importância
3. Estornar Pagamento	Prioritário	Regular	Regular	Pouca importância	Pouca importância	Pouca importância
4. Consultar Árvore de Distribuição por Docto	Prioritário	Crítico	Prioritário	Crítico	Prioritário	Prioritário
5. Consultar Distribuição Docto-Auto	Pouca importância	Prioritário	Prioritário	Prioritário	Prioritário	Prioritário
6. Consultar Distribuição por Auto	Pouca importância	Prioritário	Prioritário	Crítico	Prioritário	Prioritário
7. Gerar Perfil de Arrecadação	Prioritário	Regular	Regular	Regular	Regular	Regular
8. Auditar Pagamento sem Crédito	Regular	Pouca importância	Exceção de qualidade	Exceção de qualidade	Exceção de qualidade	Exceção de qualidade

Desta tabela devem ser cadastrados em uma planilha pré-formatada, por exemplo no excel, o qual o exemplo utilizou, todos os casos de usos com apenas um conceito, quando tiver mais de um escolher o mais alto, que será calculado automaticamente pelo excel a tabela final com resultado das prioridades. Para que se entenda como se chegou ao resultado será apresentado passo a passo. Ou seja a tabela que será entrada no o excel é a seguinte:

TABELA D.2 TABELA COM CONCEITOS REDUZIDOS PARA ENTRADA NA PLANILHA

Descrição do Caso de uso	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
1. Cadastrar Distribuição das multas	Crítico	Crítico	Pouca importância	Prioritário	Pouca importância	Crítico
2. Excluir a Distribuição	Crítico	Regular	Regular	Regular	Pouca importância	Pouca importância
3. Estornar Pagamento	Prioritário	Regular	Regular	Pouca importância	Pouca importância	Pouca importância
4. Consultar Árvore de Distribuição por Docto	Prioritário	Crítico	Prioritário	Crítico	Prioritário	Prioritário
5. Consultar Distribuição Docto-Auto	Pouca importância	Prioritário	Prioritário	Prioritário	Prioritário	Prioritário
6. Consultar Distribuição por Auto	Pouca importância	Prioritário	Prioritário	Crítico	Prioritário	Prioritário
7. Gerar Perfil de Arrecadação	Prioritário	Regular	Regular	Regular	Regular	Regular
8. Auditar Pagamento sem Crédito	Regular	Pouca importância	Exceção de qualidade	Exceção de qualidade	Exceção de qualidade	Exceção de qualidade

Com base na Tabela D.2 de conceitos de entrada para a planilha, o excel automaticamente gera uma segunda Tabela D.3, que são os valores para estes conceitos: 5 para Crítico, 4 para Prioritário, 3 para Regular, 2 para Pouca importância e 1 para Exceção de qualidade

TABELA D.3 ATRIBUIÇÃO DOS VALORES CORRESPONDENTES AOS CONCEITOS

Descrição do Caso de uso	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
1. Cadastrar Distribuição das multas	5	5	2	4	2	5
2. Excluir a Distribuição	5	3	3	3	2	2
3. Estornar Pagamento	4	3	3	2	2	2
4. Consultar Árvore de Distribuição por Docto	4	5	5	5	4	4
5. Consultar Distribuição Docto-Auto	2	4	4	4	4	4
6. Consultar Distribuição por Auto	2	4	4	5	4	4
7. Gerar Perfil de Arrecadação	4	3	3	3	3	3
8. Auditar Pagamento sem Crédito	3	2	1	1	1	1

Depois disso faz uma média dos itens 1, 2 e 3. Considerando que este itens podem não existir, faz uma média dos que existem. Se tiver dois itens divide por .2. Isto já é feito automaticamente pela planilha, mostrado na tabela D.4.

TABELA D.4 CÁLCULO DA MÉDIA DOS ITENS 1, 2 E 3

	Item 1,2,3	Item 4	Item 5	Item 6
Descrição do Caso de uso/peso	3,00	2	1	3
1. Cadastrar Distribuição das multas	4,00	4	2	5
2. Excluir a Distribuição	3,67	3	2	2
3. Estornar Pagamento	3,33	2	2	2
4. Consultar Árvore de Distribuição por Docto	4,67	5	4	4
5. Consultar Distribuição Docto-Auto	3,33	4	4	4
6. Consultar Distribuição por Auto	3,33	5	4	4
7. Gerar Perfil de Arrecadação	3,33	3	3	3
8. Auditar Pagamento sem Crédito	2,00	1	1	1

Depois multiplica os resultados pelo peso correspondente ao item, que está mostrado na segunda linha de cabeçalho, fazendo um somatório por caso de uso, resultando na Tabela D.5:

TABELA D.5 ATRIBUIÇÃO DOS PESOS CORRESPONDENTES AOS ITENS

	Item 1,2,3	Item 4	Item 5	Item 6	Somatório
Descrição do Caso de uso	peso=3	peso=2	peso=1	peso=3	
1. Cadastrar Distribuição das multas	12,00	8,00	2,00	15,00	37,00
2. Excluir a Distribuição	11,00	6,00	2,00	6,00	25,00
3. Estornar Pagamento	10,00	4,00	2,00	6,00	22,00
4. Consultar Árvore de Distribuição por Docto	14,00	10,00	4,00	12,00	40,00
5. Consultar Distribuição Docto-Auto	10,00	8,00	4,00	12,00	34,00
6. Consultar Distribuição por Auto	10,00	10,00	4,00	12,00	36,00
7. Gerar Perfil de Arrecadação	10,00	6,00	3,00	9,00	28,00
8. Auditar Pagamento sem Crédito	6,00	2,00	1,00	3,00	12,00

Com este somatório é possível identificar o conceito final para estabelecer o grau de prioridade e critério de testes a serem adotados. Se somatório for mais de 30 inclusive é alto, se for entre 20 inclusive e 30, é médio e se for menos de 20 é baixo, mostrada na tabela D.6.

TABELA D.6 ATRIBUIÇÃO DOS PESOS CORRESPONDENTES AOS ITENS

	Item 1,2,3	Item 4	Item 5	Item 6	Somatório	Conceito
Descrição do Caso de uso	**peso=3**	**peso=2**	**peso=1**	**peso=3**		**final**
1. Cadastrar Distribuição das multas	12,00	8,00	2,00	15,00	37,00	**ALTA**
2. Excluir a Distribuição	11,00	6,00	2,00	6,00	25,00	**MÉDIA**
3. Estornar Pagamento	10,00	4,00	2,00	6,00	22,00	**MÉDIA**
4. Consultar Distribuição por Docto	14,00	10,00	4,00	12,00	40,00	**ALTA**
5. Consultar Distribuição Docto-Auto	10,00	8,00	4,00	12,00	34,00	**ALTA**
6. Consultar Distribuição por Auto	10,00	10,00	4,00	12,00	36,00	**ALTA**
7. Gerar Perfil de Arrecadação	10,00	6,00	3,00	9,00	28,00	**MÉDIA**
8. Auditar Pagamento sem Crédito	6,00	2,00	1,00	3,00	12,00	**BAIXA**

Por questões de melhor visualização classifica-se a tabela por ordem da coluna somatório e se tem a tabela final com a prioridade e quão rigoroso terão que ser os critérios de teste para cada caso de uso, conforme a tabela D.7

TABELA D.7 ATRIBUIÇÃO DOS PESOS CORRESPONDENTES AOS ITENS

	Item 1,2,3	Item 4	Item 5	Item 6	Somatório	Conceito
Descrição do Caso de uso	**peso=3**	**peso=2**	**peso=1**	**peso=3**		**final**
4. Consultar Distribuição por Docto	14,00	10,00	4,00	12,00	40,00	**ALTA**
1. Cadastrar Distribuição das multas	12,00	8,00	2,00	15,00	37,00	**ALTA**
6. Consultar Distribuição por Auto	10,00	10,00	4,00	12,00	36,00	**ALTA**
5. Consultar Distribuição Docto-Auto	10,00	8,00	4,00	12,00	34,00	**ALTA**
7. Gerar Perfil de Arrecadação	10,00	6,00	3,00	9,00	28,00	**MÉDIA**
2. Excluir a Distribuição	11,00	6,00	2,00	6,00	25,00	**MÉDIA**
3. Estornar Pagamento	10,00	4,00	2,00	6,00	22,00	**MÉDIA**
8. Auditar Pagamento sem Crédito	6,00	2,00	1,00	3,00	12,00	**BAIXA**

APÊNDICE E DADOS PARA PROJETO DOS CASOS DE TESTE

Para gerar o projeto de casos de teste execução e inclusive os dados para teste de desempenho, carga e estresse são utilizados arquivos gerados a partir de dados reais contidos no *mainframe*, gerando um arquivo texto que tem o seguinte *layout*.

Id Documento: texto(16)
Autenticação: texto(48)
Tipo Documento: texto(1)
Id Órgão Arrecadação: inteiro(6)
Id Auto Infração: texto(16)
Valor Original: real (7)
Base Cálculo Original: real(7)
Id Distribuição: inteiro (9)
Data Crédito: data(8)
Valor Direito: real(7)
Valor Crédito: real(7)
Percentual Base Cálculo: real(5)
Indicativo Gerador: inteiro(1)
Código Evento Liberação: texto(1)
Id Órgão Origem: inteiro(6)
Id Órgão Destino: inteiro(6)
Id Órgão Competente: inteiro(6)
Id Distribuição Origem: inteiro(9)

Os dados gravados conforme exemplo a seguir:

Exemplos de registros (Dados de teste)

BEP-ILEGIVEL 20010503TES-116200
61162001162 00F0000135730005000000425600000328020010503000404400042561 0000G1000000116200116
2000000000000000003281000000000000212000021200500G2116200000010116200000003280
BEP-ILEGIVEL 20010507TES-116200
61162001162 00H0000153550018000001532300000464020010507001455700153231 0000G1000000116200 11
6200000000000000004641000000000000766000076600500G2116200000010116200000004640
BEP-000000 20010416TES-116200
61162001162 00D0000882040012000001021500000335920010416000970500102151 0000G1000000116200 11
6200000000000000003360000000000000510000051000500G2116200000010116200000003359
BEP-0022 20010417TES-116200
61162001162 00E0000026120018000001532300000843320010417001455700153231 0000G1000000116200116
2000000000000000084340000000000007660000766005 00G2116200000010116200000008433
BEP-002252 20010417TES-116200
61162001162 00E0000026130005000000425600000843520010417000404400042561 0000G1000000116200116
2000000000000000084360000000000021200002120050 0G2116200000010116200000008435
BEP-0030270067R 20010503TES-116200
61162001162 00D0000882100012000001021500000336320010503000970500102151 0000G1000000116200 11
6200000000000000003364000000000000510000051000500G2116200000010116200000003363
BEP-0032533178R 20010502TES-116200
61162001162 00H0000017870012000001021500000021920010502000970500102151 0000G1000000116200 11
6200000000000000000220000000000000510000051000500G2116200000010116200000000219

Se os dados forem formatados conforme o *layout* apresentado, pode-se obter informações como nos exemplos das Tabelas E.1 e E.2.

TABELA E.1 EXEMPLO 1 DE DADOS FORMATADOS PARA ELABORAÇÃO DOS CASOS DE TESTE:

Dados do documento/auto							
Documento: **2000632046716011** Data: 18/05/2001 Autenticação: 0 Tipo Documento: 2 Id Órgão Arrecadador: 116100 Id Auto Infração: **275350W000220642** Valor Original: 540.00 Base Cálculo Original: 574.61							
Dados da distribuição da distribuição do auto							
Id Distribuição:	8709	8710	8711	8712	8713	8714	8715
Data Crédito:	20010518	20010518	20010518				
Valor Direito:	0002729	0005458	0000000	0044901	0002873	0000500	0001000
Valor Crédito:	0002729	0005458	0049274	0049274	0002873	0000500	0001000
Percentual Base Cálculo:	00475	00950	08575	08575	00500	00000	00000
Indicativo Gerador:	G	G	R	R	G	A	A
Código Evento Liberação:	1	1	1	1	1	1	1
Id Órgão Origem:	000000	000000	000000	116100	275350	275350	275350
Id Órgão Destino:	116101	116103	116100	275350	000010	900000	900110
Id Órgão Competente:	275350	275350	275350	275350	275350	275350	275350
Id Distribuição Origem:	0	0	0	8711	8712	8712	8712

Para entender melhor esta árvore, acompanhe o esquema da Figura E.1:

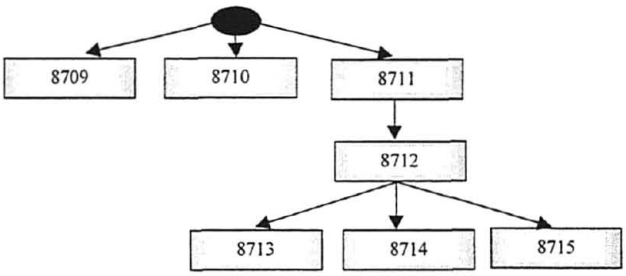


FIGURA E.1 ESQUEMA DA ÁRVORE DE DISTRIBUIÇCÃO DO EXEMPLO 1

TABELA E.2 EXEMPLO 2 DE DADOS FORMATADOS PARA ELABORAÇÃO DOS CASOS DE TESTE:

Dados do documento/auto						
Documento: **2001398308608006** Data: 28/05/2001 Autenticação: 0 Tipo Documento: 2 Id Órgão Arrecadador: 116100 Id Auto Infração: **116100E000959376** Valor Original: 50.00 Base Cálculo Original: 53.20						
Dados da distribuição da distribuição do auto						
Id Distribuição:	49217	49218	49219	49220	49221	49222
Data Crédito:	0	0	0	0	0	0
Valor Direito:	2.66	5.05	8.87	0	20.21	16.41
Valor Crédito:	2.66	5.05	8.87	36.62	20.21	16.41
Percentual Base Cálculo:	5.00	9.50	0	68.83	38.00	30.84
Indicativo Gerador:	G	G	A	R	G	R
Código Evento Liberação:	1	1	1	1	1	1
Id Órgão Origem:	0	0	0	0	116100	116100
Id Órgão Destino:	10	116101	116102	116100	900010	276910
Id Órgão Competente:	116100	116100	116100	116100	116100	116100
Id Distribuição Origem:	0	0	0	0	49220	49220

A apresentação dos dados, para os testes de aceitação devem ser conforme as Tabelas E.3, E.4 e E.5.

TABELA E.3 SAÍDA ESPERADA PARA CASO DE USO 4

GUIA	PAGO	CREDITADO	DIREITO	SITUAÇÃO
2001398308608006	R$ 106.40			
116100E000959376	R$ 53.20			
Órgão AAA=49217		R$ 2.66	R$ 2,66	Liberado
Órgão BBB=49218		R$ 5.05	R$ 5.05	Liberado
Órgão CCC=49219		R$ 8.87	R$ 8.87	Liberado
Órgão DDD=49220		R$ 36.62	R$ 0,0	Liberado
Órgão D1D1D1=49221		R$ 20.21	R$ 20.21	Liberado
Órgão D2D2D2=49222		R$ 16.41	R$ 16.41	Liberado
116100E001045102	R$ 53.20			
Órgão EEE=49223		R$ 2.66	R$ 2,66	Liberado
Órgão FFF=49224		R$ 5.05	R$ 5.05	Liberado
Órgão GGG=49225		R$ 8.87	R$ 8.87	Liberado
Órgão DDD=49226		R$ 36.62	R$ 0,0	Liberado
Órgão D1D1D1=49227		R$ 20.21	R$ 20.21	Liberado
Órgão D2D2D2=49228		R$ 16.41	R$ 16.41	Liberado
TOTAL WWW	R$ 106.40	R$ 106.40	R$ 106.40	

TABELA E.4 SAÍDA ESPERADA PARA CASO DE USO 2

GUIA	PAGO	CREDITADO	DIREITO	SITUAÇÃO
2001398308608006	R$ 106.40			
116100E000959376	R$ 36.62			
Órgão D1D1D1-49221		R$ 20.21	R$ 20.21	Liberado
Órgão D2D2D2-49222		R$ 16.41	R$ 16.41	Liberado
116100E001045102	R$ 36.62			
Órgão D1D1D1-49227		R$ 20.21	R$ 20.21	Liberado
Órgão D2D2D2-49228		R$ 16.41	R$ 16.41	Liberado
TOTAL ZZZ	R$ 73.24	R$ 73.24	R$ 73.24	

TABELA E.5 SAÍDA ESPERADA PARA CASO DE USO 3

GUIA	PAGO	CREDITADO	DIREITO	SITUAÇÃO
2001398308608006	R$ 106.40			
116100E000959376	R$ 53.20			
Órgão AAA-49217		R$ 2.66	R$ 2,66	Liberado
Órgão BBB-49218		R$ 5.05	R$ 5.05	Liberado
Órgão CCC-49219		R$ 8.87	R$ 8.87	Liberado
Órgão DDD-49220		R$ 36.62	R$ 0,0	Liberado
116100E001045102	R$ 53.20			
Órgão EEE-49223		R$ 2.66	R$ 2,66	Liberado
Órgão FFF-49224		R$ 5.05	R$ 5.05	Liberado
Órgão GGG-49225		R$ 8.87	R$ 8.87	Liberado
Órgão DDD-49226		R$ 36.62	R$ 0,0	Liberado
TOTAL WWW	R$ 106.40	R$ 106.40	R$ 106.40	

APÊNDICE F PROJETO DE CASOS DE TESTE

F.1 Caso de Uso 4 – Consultar árvore por Docto (F106)

F.1.1 Teste de Condição

TABELA F.1 CASOS DE TESTE PARA CASO DE USO 4 (F106)

Caso de Teste 1		
Nome do dado	**Dado de entrada**	**Saída esperada**
Id Docto:	2001398308608006	Auto=Distribuições: **116100E000959376**=49217,49218,49219,49220,49221,49222 e **116100E001045102**=49223,49224,49225,49226,49227,49228
Autenticação:	0	
Tipo do Docto:	2	
Período Crédito:	Nulo	
Período Pagamento:	Nulo	
Id do órgão que consulta:	**116100**	
Caso de Teste 2		
Nome do dado	**Dado de entrada**	**Saída esperada**
Id Docto:	2001398308608006	Auto=Distribuições: **116100E000959376**=49221,49222 e **116100E001045102**=49227,49228
Autenticação:	0	
Tipo do Docto:	2	
Período Crédito:	Nulo	
Período Pagamento:	Nulo	
Id do órgão que consulta:	900010	
Caso de Teste 3		
Nome do dado	**Dado de entrada**	**Saída esperada**
Id Docto:	2001398308608006	Auto=Distribuições: **116100E000959376**=49217,49218,49219,49220 **116100E001045102**=49223,49224,49225,49226
Autenticação:	0	
Tipo do Docto:	2	
Período Crédito:	Nulo	
Período Pagamento:	Nulo	
Id do órgão que consulta:	**116101**	
Caso de Teste 4		
Nome do dado	**Dado de entrada**	**Saída esperada**
Id Docto:	2000632046716011	Auto=Distribuições: **275350W000220642**=8709,8710,8711,8712 **116100E000547813**=8716,8717,8718,8719,8720,8721 **275350W000277105**=8722,8723,8724,8725 **275350W000401196**=8729,8730,8731,8732
Autenticação:	0	
Tipo do Docto:	2	
Período Crédito:	Nulo	
Período Pagamento:	Nulo	
Id do órgão que consulta:	**116100**	
Caso de Teste 5		
Nome do dado	**Dado de entrada**	**Saída esperada**
Id Docto:	2000632046716011	Auto =Distribuições: **275350W000220642**=8709,8710,8711,8712,8713,8714,8715 **275350W000277105**=8722,8723,8724,8725,8726,8727,8728 **275350W000401196**=8729,8730,8731,8732,8733,8734,8735
Autenticação:	0	
Tipo do Docto:	2	
Período Crédito:	Nulo	
Período Pagamento:	Nulo	
Id do órgão que consulta:	**275350**	
Caso de Teste 6		
Nome do dado	**Dado de entrada**	**Saída esperada**
Id Docto:	2000632046716011	Auto=Distribuições: **275350W000220642**=8709,8710,8711 **275350W000277105**=8722,8723,8724 **275350W000401196**=8729,8730,8731
Autenticação:	0	
Tipo do Docto:	2	
Período Crédito:	**18/05/2001**	
Período Pagamento:	Nulo	
Id do órgão que consulta:	**275350**	

F.1.2 Teste de segurança

Supondo uma arquitetura semelhante e simplificada da real, deve ser procurado testar a arquitetura de um sistema complexo, principalmente no aspecto segurança. Observando detalhes que devam ser implementados para que sejam respeitadas essas questões de segurança. Estes testes devem ser feitos antes mesmo do início do sistema, ou após implementar parte de 1 ou 2 casos mais simples, para validar este aspecto de segurança na arquitetura.

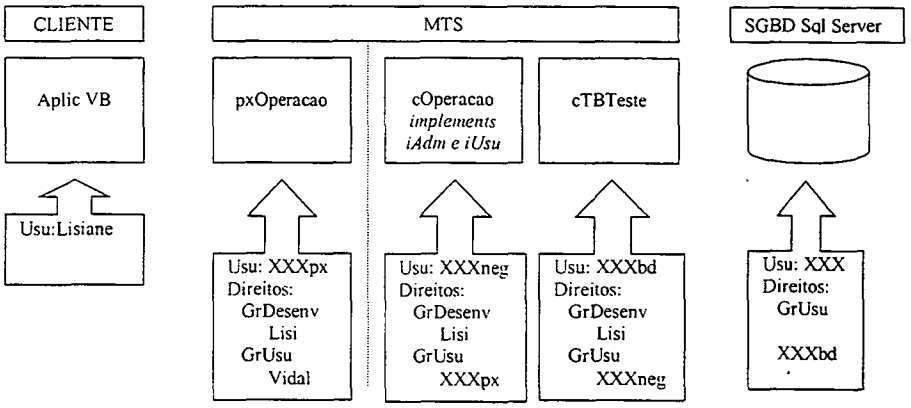


FIGURA F.1 ARQUITETURA SIMPLIFICADA PARA O TESTE DE SEGURANÇA

Para a arquitetura simplificada da Figura F.1 executar os casos de teste conforme a Tabela F.2.

TABELA F.2 CASOS DE TESTE PARA TESTE DE SEGURANÇA

Dados de entrada	Condições	Resultado esperado
Usuário: Lisiane Senha: xxx	Tentar logar com senha errada	Senha incorreta
	Tentar executar componente pxOperacao	Ok
	Tentar executar componente cTBTeste	Ok
	Tentar entrar diretamente na base Sql Server	Acesso negado
Usuário: Furquim Senha: zzz	Tentar logar com senha errada	Senha incorreta
	Tentar executar componente pxOperacao	Acesso negado
	Tentar executar componente cTBTeste	Acesso negado
	Tentar entrar diretamente na base Sql Server	Acesso negado
Usuário: Vidal Senha: zzz	Tentar logar com senha errada	Senha incorreta
	Tentar executar componente pxOperacao	Ok
	Tentar executar componente cTBTeste	Acesso negado
	Tentar entrar diretamente na base Sql Server	Acesso negado

F.1.3 Teste de estresse

A partir de um arquivo *batch* texto gerado pelo *mainframe* com 10.740 registros foi feita uma carga na tabela de distribuição com 50.000 registros, e feito um programa para controlar os tempos com 10 usuários simultâneos acessar os componentes automaticamente por 1 hora e meia direto.

Dados de entrada

Uma descrição do arquivo utilizado com *layout* e exemplo de dados está relacionado no APÊNDICE E, na Seção E.3.

Resultado esperado

Que todas as consultas sejam executadas sem problema e sem travar o componente nem o servidor.

F.1.4 Teste de desempenho

Como o caso de uso 4 – Consultar Árvore por Docto, a característica básica são acessos, e o fator desempenho é primordial, por isso o esforço maior do teste deve ser concentrado nesta tipo de teste.

Dados de entrada

Uma descrição do arquivo utilizado com *layout* e exemplo de dados está relacionado no APÊNDICE E, na Seção E.3.

Resultado esperado

Tem consultas com 3 a 24 distribuições. É esperado que 90% dos casos atinjam um tempo de 1 segundo para as menores e no máximo 5 segundos para as maiores, com 1 ou vários usuários executando o mesmo caso de uso.

F.2 Caso de Uso 1 – Cadastrar Árvore de Distribuição (F101)

F.2.1 Teste de integridade

Este caso de uso atualiza 4 tabelas e deve fazer parte de uma transação. Primeiro atualiza a tabela documentos com os 4 primeiros dados, depois uma tabela de Autos do documento, com Id auto infração, valor original e base de cálculo, depois uma tabela de relação das duas, e posteriormente cadastra uma tabela de auto relacionamento com a árvore de distribuição.

Para um primeiro teste será utilizado dados corretos, depois um item errado com tipo de documento inexistente e outro teste cadastrando item duplicado na tabela de autos do documento.

- **Dados de teste para caso de teste 1** (Item errado)

  Documento: 11610000618393

  Data: 18/05/2001

  *Autenticação: BPR3792528005R180501 GRM 000313**********85,12*

  ***Tipo Documento:** 5 seria o certo mas o teste deve incluir o documento tipo 9 que não existe*

Id Órgão Arrecadador: 116100

Id Auto Infração: 116100E000428712

Valor Original: 80.00
Base Cálculo Original: 85.12

Com a seguinte árvore de distribuição:

Id Distribuição:	000006689	000006690	000006691	000006692	000006693	000006694
Data Crédito:	00000000	00000000	00000000	00000000	00000000	00000000
Valor Direito:	0000425	0000808	0000887	0000000	0003234	0003158
Valor Crédito:	0000425	0000808	0000887	0006392	0003234	0003158
Percentual Base Cálculo:	00500	00950	00000	07509	03800	03710
Indicativo Gerador:	G	G	A	R	G	R
Código Evento Liberação:	1	1	1	1	1	1
Id Órgão Origem:	000000	000000	000000	000000	116100	116100
Id Órgão Destino:	000010	116101	116102	116100	900010	277790
Id Órgão Competente:	116100	116100	116100	116100	116100	116100
Id Distribuição Origem:	000000000	000000000	000000000	000000000	000006692	000006692

Resultado esperado: exclua o documento e nem cadastre o restante

- **Dados de teste para caso de teste 2** (Item errado)

Valores possíveis para indicativo gerador: G, A, R (X não existe)

Documento: 11610000618393

Data: 18/05/2001

*Autenticação: BPR3792528005R180501 GRM 000313**********85,12*

Tipo Documento: 5

Id Órgão Arrecadador: 116100

Id Auto Infração: 116100E000428712

Valor Original: 80.00

Base Cálculo Original: 85.12

Com a seguinte árvore de distribuição:

Id Distribuição:	000006689	000006690	000006691	000006692	000006693	000006694
Data Crédito:	00000000	00000000	00000000	00000000	00000000	00000000
Valor Direito:	0000425	0000808	0000887	0000000	0003234	0003158
Valor Crédito:	0000425	0000808	0000887	0006392	0003234	0003158
Percentual Base Cálculo:	00500	00950	00000	07509	03800	03710
Indicativo Gerador:	G	G	A	R	***X não existe***	R
Código Evento Liberação:	1	1	1	1	1	1
Id Órgão Origem:	000000	000000	000000	000000	116100	116100
Id Órgão Destino:	000010	116101	116102	116100	900010	277790
Id Órgão Competente:	116100	116100	116100	116100	116100	116100
Id Distribuição Origem:	000000000	000000000	000000000	000000000	000006692	000006692

Resultado esperado: exclua o documento, o auto de infração deste documento e as distribuições 6689, 6690, 6691, 6692 que já tinham sido incluídas e nem cadastre o restante.

- **Dados de teste para caso de teste 3** (Item correto)

  Documento: 11610000618393

  Data: 18/05/2001

  *Autenticação: BPR3792528005R180501 GRM 000313**********85,12*

  Tipo Documento: 5

  Id Órgão Arrecadador: 116100

  Id Auto Infração: 116100E000428712

  Valor Original: 80.00

  Base Cálculo Original: 85.12

Com a seguinte árvore de distribuição:

Id Distribuição:	000006689	000006690	000006691	000006692	000006693	000006694
Data Crédito:	00000000	00000000	00000000	00000000	00000000	00000000
Valor Direito:	0000425	0000808	0000887	0000000	0003234	0003158
Valor Crédito:	0000425	0000808	0000887	0006392	0003234	0003158
Percentual Base Cálculo:	00500	00950	00000	07509	03800	03710
Indicativo Gerador:	G	G	A	R	G	R
Código Evento Liberação:	1	1	1	1	1	1
Id Órgão Origem:	000000	000000	000000	000000	116100	116100
Id Órgão Destino:	000010	116101	116102	116100	900010	277790
Id Órgão Competente:	116100	116100	116100	116100	116100	116100
Id Distribuição Origem:	000000000	000000000	000000000	000000000	000006692	000006692

Resultado esperado: inclua todos os registros

APÊNDICE G EXECUÇÃO DOS CASOS DE TESTE

G.1 Caso de Uso 4 – Consultar Árvore de Distribuição por Docto (F106)

G.1.1 Teste de condição

Caso de Teste 1			
Nome do dado	**Dado de entrada**	**Saída esperada**	**Resultado obtido**
Id Docto:	2001398308608006	Auto=Distribuições: **116100E000959376**=49217,49218,49219,49220,49221,49222 **116100E001045102**=49223,49224,49225,49226,49227,49228	Auto=Distribuições: **116100E000959376**=49217,49218,49219,49220,49221,49222 **116100E001045102**=49223,49224,49225,49226,49227,49228
Autenticação:	0		
Tipo do Docto:	2		
Período Crédito:	Nulo		
Período Pagamento:	Nulo		
Id do órgão que consulta:	116100		
Caso de Teste 2			
Nome do dado	**Dado de entrada**	**Saída esperada**	**Resultado obtido**
Id Docto:	2001398308608006	Auto=Distribuições: **116100E000959376**=49221,49222 **116100E001045102**=49227,49228	Auto=Distribuições: **116100E000959376**=49221,49222 **116100E001045102**=49227,49228
Autenticação:	0		
Tipo do Docto:	2		
Período Crédito:	Nulo		
Período Pagamento:	Nulo		
Id do órgão que consulta:	900010		
Caso de Teste 3			
Nome do dado	**Dado de entrada**	**Saída esperada**	**Resultado obtido**
Id Docto:	2001398308608006	Auto=Distribuições: **116100E000959376**=49217,49218,49219,49220 **116100E001045102**=49223,49224,49225,49226	Auto=Distribuições: **116100E000959376**=49221
Autenticação:	0		
Tipo do Docto:	2		
Período Crédito:	Nulo		
Período Pagamento:	Nulo		
Id do órgão que consulta:	116101		
Caso de Teste 4			
Nome do dado	**Dado de entrada**	**Saída esperada**	**Resultado obtido**
Id Docto:	2000632046716011	Auto=Distribuições: **275350W000220642**=8709,8710,8711,8712 **116100E000547813**=8716,8717,8718,8719,8720,8721 **275350W000277105**=8722,8723,8724,8725 **275350W000401196**=8729,8730,8731,8732	Auto=Distribuições: **275350W000220642**=8709,8710,8711,8712 **116100E000547813**=8716,8717,8718,8719,8720,8721 **275350W000277105**=8722,8723,8724,8725 **275350W000401196**=8729,8730,8731,8732
Autenticação:	0		
Tipo do Docto:	2		
Período Crédito:	Nulo		
Período Pagamento:	Nulo		
Id do órgão que consulta:	116100		

Caso de Teste 5			
Nome do dado	**Dado de entrada**	**Saída esperada**	**Resultado obtido**
Id Docto:	2000632046716011	Auto =Distribuições: **275350W000220642**=8709,8710,8711,8712,8713,8714,8715 **275350W000277105**=8722,8723,8724,8725,8726,8727,8728 **275350W000401196**=8729,8730,8731,8732,8733,8734,8735	Auto=Distribuições: **275350W000220642**=8709,8710,8711,8712,8713,8714,87151 **16100E000547813**=8719,8720,8721 **275350W000277105**=8722,8723,8724,8725,8726,8727,8728 **275350W000401196**=8729,8730,8731,8732,8733,8734,8735
Autenticação:	0		
Tipo do Docto:	2		
Período Crédito:	Nulo		
Período Pagamento:	Nulo		
Id do órgão que consulta:	275350		

G.1.2 Teste de segurança

Na execução dos casos de teste para o teste de segurança, pode ser identificado que ocorreu um resultado obtido diferente do que era esperado. Uma falha cujas causas eram provenientes de configuração do Monitor de transação que são explicadas na seqüência.

TABELA G.1 RESULTADO OBTIDO DA APLICAÇÃO DO TESTE DE SEGURANÇA

Dados de entrada	Ações executadas	Resultado esperado	Resultado obtido
Login Lisiane/senha xxx	Criação do componente pxOperacao e Execução do método do componente pxOperacao	Ok	Ok
Login Lisiane/senha xxx	Criação do componente cTBTeste e Execução do método do componente cTBTeste	Ok	Ok
Login Lisiane/senha xxx	Conectar diretamente na base X e Executar as stored procedures	Permissão negada	Permissão negada
Login Furquim/senha yyy	Criação do componente pxOperacao e Execução do método do componente pxOperacao	Permissão negada	Permissão Negada
Login Vida/senha zzz	Criação do componente pxOperacao e Execução do método do componente pxOperacao	Ok	Permissão negado
Login Vida/senha zzz	Criação do componente cTBTeste e Execução do método do componente cTBTeste	Permissão negada	Permissão negada
Login Vidal/senha zzz	Conectar diretamente na base X e Executar as stored procedures	Permissão negada	Permissão negada

Para se conseguir executar os testes de segurança foram identificadas algumas informações importantes para a criação dos pacotes no MTS:

- No MTS, na propriedade de cada pacote, guia de *security*, marcar a opção de "*enforce access check for this application*", na guia *identity*, setar opção "*this user*" e especificar usuário. Detalhe, este usuário deve ser um *login* válido no NT, os que vão rodar só no MTS, pode ser um usuário de domínio só da máquina como é o caso do Xpx, Xneg, mas o Xbd vai chegar no *Sql Server* que está em outro servidor por isso tem que estar em um domínio que possa ser visto pelo SGBD e MTS, nesse caso no domínio Y. Quando for especificado o Xbd, não esquecer de colocar o domínio Y na frente, fica "Y\Xbd". Caso alguém que não tem direito tente acessar vai aparecer um *"RunTime error '70' Permission denied"*.
- No MTS, devem ser criadas *Roles* ou Regras de acesso para aquele pacote. Define-se regras, por exemplo grUsu e grDesenv e GrAdm, e inclui usuários para cada regra. Para cada componente, na guia de *Security*, marcar a opção "*enforce component level access checks*", se não marcar esta opção vale o acesso do pacote.

Com esta opção marcada deve-se marcar as regras que poderão executar este componente.

- No MTS, na guia de *Activation*, deve ser setado se é um pacote do tipo server ou do tipo *library*. Observando que pacote *server* roda num processo separado, com o usuário especificado ou o que tiver logado, no *library* vai sempre rodar no processo que chamou e com o usuário de quem chama.
- Para que possam ser validados os usuários do NT no *sql server*, além de ter um usuário do domínio Y, o SGBD tem que estar configurado para aceitar usuários *windows* (segurança integrada) do NT e *sql server.*

G.1.3 Teste de estresse

Dados de entrada

Uma descrição do arquivo utilizado com *layout* e exemplo de dados está relacionado no APÊNDICE E, na Sub-seção E.3.

Resultado esperado

Que todas as consultas sejam executadas sem problema e sem travar o componente nem o servidor.

Resultado obtido

Quando n usuários estavam executando os componentes, algumas máquinas/usuários apresentaram um tipo de erro: "*3704 – Operation is not allowed when the object is closed*" enquanto outras (mais de uma) continuavam executando os mesmos componentes normalmente. É necessário encontrar a razão do erro.

G.1.4 Teste de desempenho

Como o caso de uso 4 – Consultar Árvore por Docto, a característica básica são acessos, e o fator desempenho é primordial, por isso o esforço maior do teste deve ser concentrado nesta tipo de teste.

Dados de entrada

Uma descrição do arquivo utilizado com *layout* e exemplo de dados está relacionado no APÊNDICE E, na Seção E.3.

Resultado esperado

Tem consultas com 3 a 24 distribuições. É esperado um tempo de 1 segundo para as menores e no máximo 5 segundos para as maiores, com 1 ou vários usuários acessando.

Resultado obtido

Conforme observado na Tabela G.2, é calculado o tempo de chamadas das funções e outro tempo considerando a apresentação do resultado em tela inclusive; também é calculado o número de distribuições que retorna para cada consulta e também o número de bytes que retornou. Os resultados dos tempos são gravados em arquivo texto, e posteriormente, foram agrupados por função, por número de máquinas executou o teste. A análise mais justa é feita considerando a mesma máquina, pois desconsidera aspectos de ambiente de máquina.

TABELA G.2 MÉDIA DO TEMPO DO TESTE DE DESEMPENHO

Maq	QtdMq	Função	CS3-D			
			Bytes	Regs	TpCmp	TpTran
03Dante	1	F105	514,84	4,5	1,819	4,193
		F106	2098,28	18,13	1,735	10,827
		F109	495,31	4,31	1,606	4,003
		Todas	1108,33	4953,02	1,721	6,646
	11	F105	500,64	4,36	3,827	14,800
		F106	620,33	5,33	1,582	16,123
		F109	579,73	5	0,896	14,289
		Todas	551,87	1016	2,353	14,875
03Marcia	1	F105	591,27	5,14	2,725	5,420
		F106	2355,55	20,55	1,839	13,453
		F109	606,76	5,294	0,697	4,387
		Todas	1153,03	4028	1,840	7,649
	11	F105	565,87	4,87	2,669	12,21
		F106	2313,90	20,18	5,100	46,265
		F109	573,31	5	1,588	11,368
		Todas	1169,78	4015,5	3,066	23,574
04Furca	1	F105	521,22	4,55	2,309	6,921
		F106	1155,72	10	1,822	12,504
		F109	512,9	4,4	1,154	6,498
		Todas	751,1	4514,5	1,746	8,827
	11	F105	263,63	2,26	3,509	6,196
		F106	1538,95	13,45	2,341	32,698
		F109	278,07	2,43	0,939	4,226
		Todas	777,44	850,56	2,388	16,294

G.2 Caso de Uso 1 – Cadastrar Árvore de Distribuição (F101)

G.2.1 Teste de integridade

Todos os resultados esperados foram obtidos.

- **Dados de teste** (Item errado)

  Documento: 11610000618393

  Data: 18/05/2001

  *Autenticação: BPR3792528005R180501 GRM 000313**********85,12*

***Tipo Documento**: 5 seria o certo mas o teste deve incluir o documento tipo 9 que não existe*

Id Órgão Arrecadador: 116100

Id Auto Infração: 116100E000428712

Valor Original: 80.00

Base Cálculo Original: 85.12

Resultado esperado: exclua o documento e nem cadastre o restante

Resultado obtido: documento foi excluído e nem cadastrado o restante

- **Dados de teste** (Item errado)

Valores possíveis para indicativo gerador: G, A, R (X não existe)

Documento: 11610000618393

Data: 18/05/2001

*Autenticação: BPR3792528005R180501 GRM 000313**********85,12*

Tipo Documento: 5

Id Órgão Arrecadador: 116100

Id Auto Infração: 116100E000428712

Valor Original: 80.00

Base Cálculo Original: 85.12

Resultado esperado: sejam excluídos o documento, o auto de infração deste documento e as distribuições 6689, 6690, 6691, 6692 que já tinham sido incluídas e nem cadastre o restante.

Resultado obtido: igual ao esperado

- **Dados de teste** (Item correto)

Documento: 11610000618393

Data: 18/05/2001

*Autenticação: BPR3792528005R180501 GRM 000313**********85,12*

Tipo Documento: 5

Id Órgão Arrecadador: 116100

Id Auto Infração: 116100E000428712

Valor Original: 80.00

Base Cálculo Original: 85.12

Resultado esperado: inclua todos os registros

Resultado obtido: todos os registros foram incluídos corretamente

APÊNDICE H ARQUITETURAS UTILIZADAS NOS TESTES

Para a execução do teste de desempenho dos componentes foi necessário variar a arquitetura. No texto tem uma explicação sucinta das arquiteturas e neste apêndice um desenho que ajuda na compreensão das mesmas.

H.1 Diagrama de Execução para Arquiteturas CS2

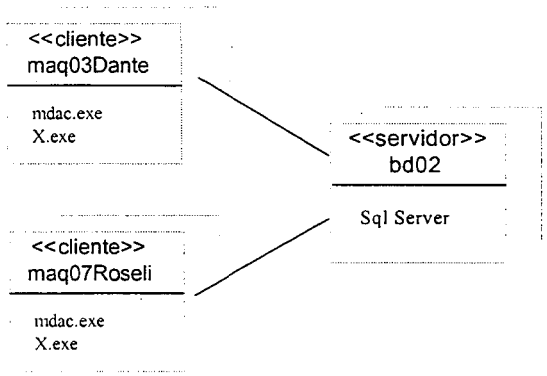


FIGURA H.1 DIAGRAMA DE COMPONENTES DA ARQUITETURA CLIENTE-SERVIDOR 2 CAMADAS

H.2 Diagrama de Execução para Arquiteturas CS3M e CS3D

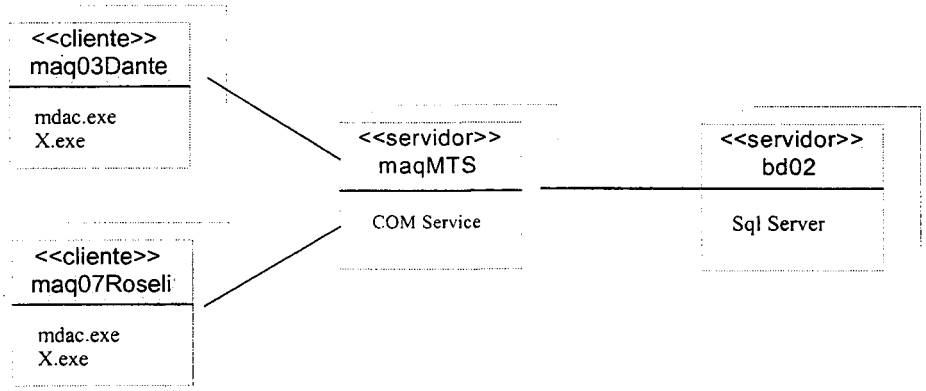


FIGURA H.2 DIAGRAMA DE EXECUÇÃO DA ARQUITETURA CLIENTE-SERVIDOR 3 CAMADAS

H.3 Diagrama de Componentes para Arquitetura CS3M

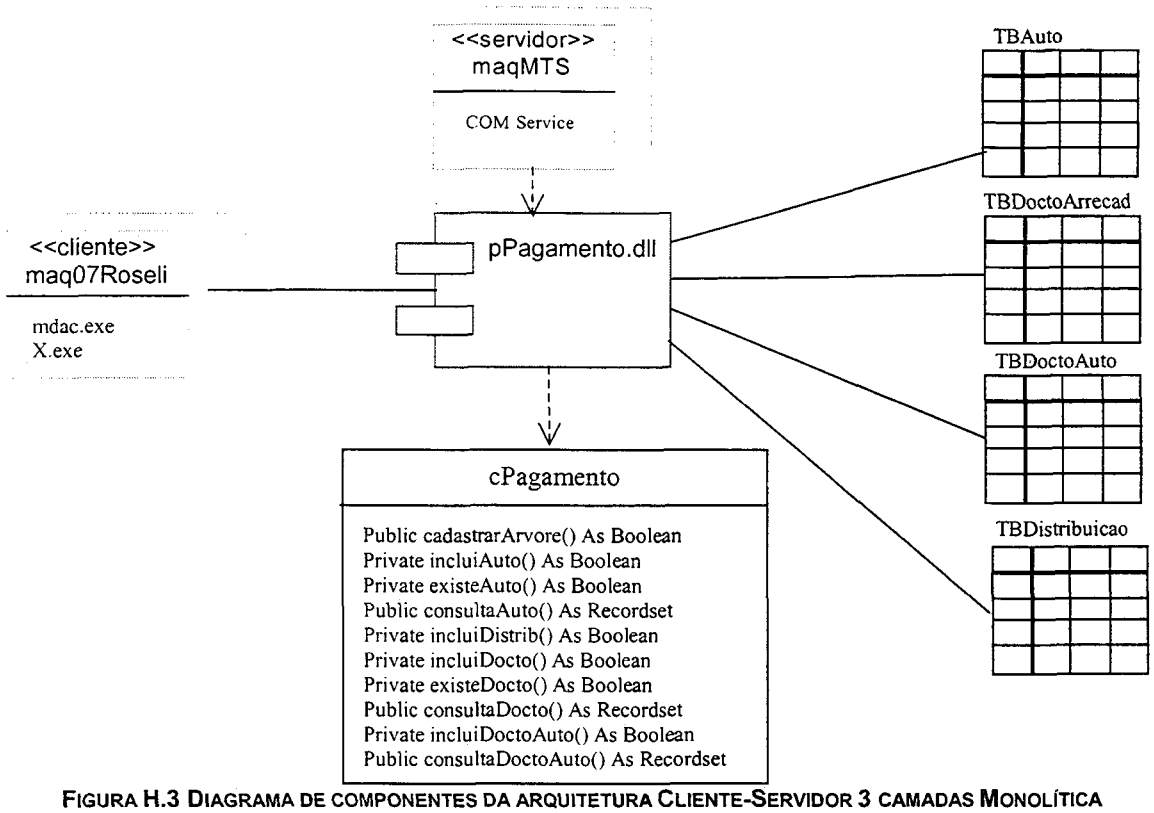


FIGURA H.3 DIAGRAMA DE COMPONENTES DA ARQUITETURA CLIENTE-SERVIDOR 3 CAMADAS MONOLÍTICA

H.4 Diagrama de Componentes para Arquitetura CS3D com thread

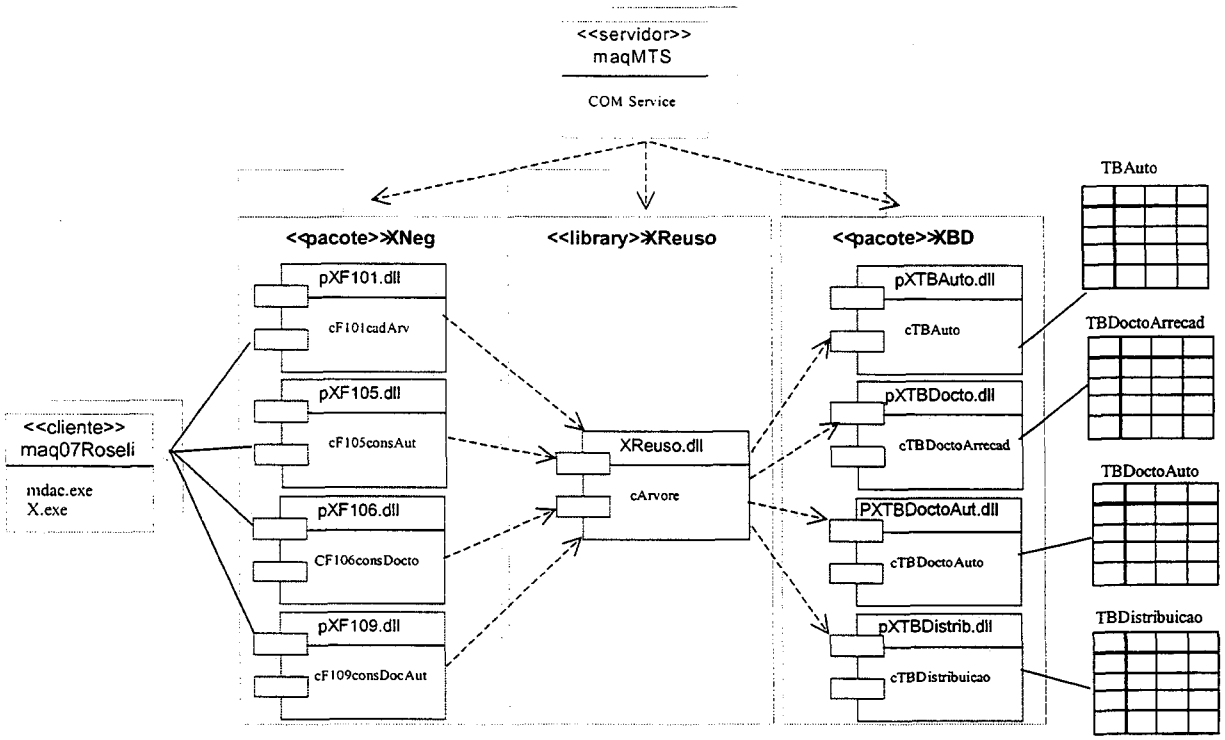


FIGURA H.1 DIAGRAMA DE COMPONENTES DA ARQUITETURA CLIENTE-SERVIDOR 3 CAMADAS DISTRIBUÍDA

ANEXO 1 DETALHAMENTO DOS CASOS DE USOS UTILIZADOS

2.1 F101 – Cadastrar Árvore de Distribuição

Ações

1. Este caso de uso inicia com o recebimento de uma árvore de distribuição de pagamento.
2. Cadastrar as informações do auto e a relação entre o auto e o documento de arrecadação.
3. Cadastrar as distribuições de pagamento pertencentes à árvore.

Informações Adicionais

A árvore será cadastrada com situação "calculada" e a atualização da situação será feita através dos casos de uso que tratam de crédito, repasse e compensação.
As árvores de distribuição de pagamento e as respectivas compensações somente serão liberadas após o recebimento das informações de crédito correspondentes e de acordo com o código do evento de liberação.

2.2 F102 – Excluir Árvore de Distribuição

Pré-Condições

Não existir nenhum crédito para a árvore.

Ações

1. Verificar se existe algum outro pagamento com data posterior à do pagamento cuja árvore de distribuição está sendo excluída. Se existir, gerar um "log" de aviso indicando que pagamentos posteriores do auto conterão distribuições erradas.
2. Excluir a árvore de distribuição e as compensações correspondentes.

Informações Adicionais

Se houver pagamento de auto com data posterior à do pagamento cuja árvore está sendo excluída, suas distribuições estarão erradas, pois terão levado em conta a existência da árvore que está sendo excluída. Nesse caso, os órgãos deverão fazer compensações entre si para ajustar os valores de direito de recebimento.
O usuário terá acesso aos avisos gerados por meio do painel de controle.

Referências Cruzadas

DTE Distribuição do Pagamento.
F001 – Painel de Controle.

2.3 F104 – Estornar Pagamento

Pré-condições

A solicitação de estorno do pagamento deve ocorrer entre D+1 e D+4, período em que o crédito correspondente ao pagamento ainda não foi efetivado e, portanto, não houve nenhum repasse de recursos financeiros.

Ações

1. Se a árvore de distribuição de pagamento contiver um nó em que o órgão de direito de recebimento é o convênio especial compensação-FUNSET-competente, executar F416 – Registrar Lançamento Compensação Funset. O tipo da operação será "Estorno de Pagamento".
2. F102 – Excluir Árvore de Distribuição

Informações Adicionais

A presença de uma distribuição de pagamento para o convênio especial ***compensação-FUNSET-competente*** na árvore que está sendo estornada indica que a dívida da FUNSET com o órgão competente em questão foi reduzida no momento da geração árvore, pois havia uma expectativa de entrada de dinheiro. Diante do pedido de estorno, essa expectativa não se confirma, fato que justifica o lançamento de compensação com FUNSET, retornando a dívida ao seu estado anterior.

A função F416 – Registrar Lançamento Compensação Funset gerará os lançamentos necessários para o sistema Y-*Mainframe*.

O recebimento do crédito implica repasse automático para as distribuições de pagamento que se encontram no primeiro nível da árvore.

Referências Cruzadas

F102 – Excluir Árvore de Distribuição – rotina externa.

F416 – Registrar Lançamento Compensação Funset – rotina externa.

2.4 F105 – Consultar Árvore de Distribuição por Auto

Ações

1. usuário informa o auto a consultar.
2. Para cada documento de arrecadação em que o auto está presente:
3. F109 – Consultar Distribuição Docto-Auto.
4. Calcular o valor pago do auto a partir dos documentos de arrecadação nos quais ele está presente.
5. Se o órgão que está fazendo a consulta não for conveniado, não se deve apresentar o valor total pago, calculado no passo anterior.

6. Mostrar as árvores de distribuição de pagamento, permitindo a navegação do usuário através dos documentos de arrecadação (guias).

Informações Adicionais

Os órgãos executivos de trânsito poderão visualizar completamente as árvores de distribuição de pagamento em que estiverem presentes. Os demais verão apenas os valores que irão receber e todos os valores subordinados a esses nas respectivas árvores.

Esboço da Interface

Modelo de apresentação das árvores de distribuição de pagamento para o órgão competente ABC:

Auto: 275350-Z00001234

Pago: R$ 200,00

DOCTO/ÓRGÃO	PAGO	CREDITADO	DIREITO	SITUAÇÃO
GRD 12344	R$ 100,			
GRD 12345	R$ 100,			
FUNSET		R$ 5,00	R$ 5,00	Repassado
XYZ-DOCTO		R$ 10,00	R$ 10,00	Repassado
XYZ-CAD		R$ 10,00	R$ 10,00	Repassado
CDEFG		R$ 5,00	R$ 5,00	Repassado
ABC		**R$ 70,00**	**R$ 50,00**	**Repassado**
PERKONS		R$ 0,00	R$ 20,00	Liberado

2.5 F106 – Consultar Árvore de Distribuição por Docto

Pré Condição

Ser órgão executivo de trânsito.

Ações

1. O usuário informa os critérios para a seleção dos documentos que serão consultadas: número do documento de arrecadação; ou período de pagamento, período de crédito, tipo do documento.
2. O sistema seleciona os documentos de arrecadação que satisfazem os critérios de seleção informados pelo usuário.
3. Para cada documento de arrecadação selecionado no passo anterior:
4. Filtrar os autos de infração em que está presente o órgão que solicitou a consulta.
5. Para cada auto filtrado
6. F109 – Consultar Distribuição Docto-Auto.
7. Considerando apenas os autos selecionados, totalizar o valor pago, o valor de direito e o valor creditado para o órgão que solicitou a consulta.
8. Se foi informado um documento, mostrar seus autos e permitir que o usuário os expanda para ver as respectivas árvores de distribuição de pagamento.

9. Se foram selecionados vários documentos, mostrá-los e permitir que o usuário as expanda para ver seus autos. Permitir também a expansão dos autos para visualizar as respectivas árvores de distribuição de pagamento.

Informações Adicionais

O órgão arrecadador (dono do documento) poderá visualizar todas as distribuições dos autos contidos no documento.

Os órgãos executivos de trânsito poderão visualizar completamente as árvores de distribuição de pagamento em que estiverem presentes. Os demais verão apenas os valores que irão receber e todos os valores subordinados a esses nas respectivas árvores.

Esboço da Interface

Modelo de apresentação da árvore de distribuição, cujo Órgão Competente é a ABC:

DOCTO	PAGO	CREDITADO	DIREITO	SITUAÇÃO
GRD 12340	R$ 120,			
GRD 12341	R$ 220,			
GRD 12342	R$ 120,			
GRD 12343	R$ 300,			
116100-A00001200	R$ 100,			
116100-C00001211	R$ 100,			
275350-Z00001222	R$ 100,			
FUNSET		R$ 5,	R$ 5	Repassado
XYZ-DOCTO		R$ 10,	R$ 10,	Repassado
XYZ-CAD		R$ 10,	R$ 10,	Repassado
CDEFG		R$ 5,	R$ 5,	Repassado
ABC		**R$ 70,**	**R$ 50,**	**Repassado**
PERKONS		R$ 0,	R$ 20,	Liberado
GRD 12344	R$ 50,			
GRD 12345	R$ 150,			
TOTAL ABC	R$ 960,	R$ 580,	R$ 450,	

2.6 F107 – Gerar Perfil de Arrecadação

Ações

1. usuário informa o convênio e o período desejado (data inicial e final).
2. Para o perfil do que foi arrecadado, acessar as Distribuições de Pagamento do convênio com data de crédito dentro do intervalo solicitado, totalizando os valores para cada tipo de documento.
3. Para o perfil do que está previsto, acessar as Distribuições de Pagamento do convênio cujo crédito ainda não foi efetivado, totalizando os valores para cada tipo de documento.
4. Totalizar os valores e mostrar o perfil.

Esboço da Interface

A seguir, esboço do Perfil de Arrecadação, considerando o convênio DER/PR.

Arrecadado

Data Crédito	Tipo Doc	Qtde Doc	Qtde Autos	Vlr Creditado	Vlr Direito
21.03.00	GRM	13	13	138,00	130,00
	GRU	10	12	320,00	200,00
	Espontâneo	11	11	235,00	120,00
22.03.00	GRD	33	33	567,00	320,00
	GRM	13	14	543,00	321,00
	GRLAV	13	14	543,00	321,00
	GRU	10	14	343,00	221,00
	Espontâneo	34	34	533,00	121,00
23.03.00	GRD	63	63	1245,00	800,00
	GRM	3	5	45,00	14,00
	GRU	3	5	54,00	24,00
	Espontâneo	68	68	3221,00	1234,00
TOTAL PERÍODO		**274**	**286**	**7787,00**	**3826,00**

Previsto

Data Pagamento	Tipo Documento	Qtde Doc	Qtde Autos	Vlr Previsto	Vlr Direito
15.07.00	GRM	13	13	138,00	130,00
	GRU	10	12	320,00	200,00
	Espontâneo	11	11	235,00	120,00
16.07.00	GRD	63	63	1245,00	800,00
	GRM	3	5	45,00	14,00
	GRU	3	5	54,00	24,00
	Espontâneo	68	68	3221,00	1234,00
TOTAL PERÍODO		**171**	**177**	**5258,00**	**2522,00**

2.7 F108 – Auditar Pagamento sem Crédito

Ações

1. Este caso de uso inicia com a solicitação de auditoria pelo usuário. O usuário informa o número de dias esperado entre o pagamento de um documento de arrecadação e a efetivação do respectivo crédito pelo banco. Se ele não informar nada, será adotado o número 5 como padrão.
2. Para cada documento que possua informação de pagamento e não possua informação de crédito, deve-se realizar a seguinte operação:
   2.1. Se o número de dias decorridos entre a data do pagamento e a data atual for maior que o número de dias informado no passo 2, então listar o documento.
3. Ao final, listar o total de documentos sem crédito.

Informações Adicionais

Atualmente, existe um convênio com o banco, especificando que o crédito deve ocorrer no máximo 4 dias após o pagamento de um documento de arrecadação. Assim, pagamentos que permaneçam sem crédito por mais do que 4 dias estão em situação irregular e devem gerar um alerta para o usuário. Devido à existência desse convênio, caso o usuário não informe o número de dias esperado entre o pagamento e o crédito, o sistema X usará como *default* o valor 5.

O software X sabe que um documento de arrecadação foi pago pela existência de uma árvore de distribuição correspondente.

2.8 F109 – Consultar Distribuição Docto-Auto

Pré-condições

Conhecer o órgão que está realizando a consulta.

Conhecer o documento consultado.

Conhecer o auto consultado.

Ações

1. Este caso de uso inicia com o recebimento da solicitação de consulta, contendo o órgão que está pesquisando as informações de distribuição, a identificação do documento de arrecadação e a identificação do auto.
2. Filtrar as distribuições de pagamento que serão apresentadas na consulta de acordo com a situação do órgão (arrecadador ou não; órgão executivo de trânsito ou não).
3. Recuperar o valor de direito, o valor creditado e a situação da distribuição de pagamento.

Informações Adicionais

O órgão arrecadador (dono do documento) poderá visualizar todas as distribuições dos autos contidos no documento.

Os órgãos executivos de trânsito poderão visualizar completamente as árvores de distribuição de pagamento em que estiverem presentes. Os demais verão apenas os valores que irão receber e todos os valores subordinados a esses nas respectivas árvores.