FireWeb: Uma Ferramenta de Suporte aos Testes de Regressão de Aplicações Web

Wennder Indalécio Oliveira Fidelis

Trabalho Final de Mestrado Profissional

Instituto de Computação
Universidade Estadual de Campinas

FireWeb: Uma Ferramenta de Suporte aos Testes de Regressão de Aplicações Web

Wennder Indalécio Oliveira Fidelis

Dezembro / 2003

Banca Examinadora:

- Profa. Dra. Eliane Martins (Orientadora)
  Instituto de Computação – UNICAMP
- Profa. Dra. Maria Cecília Calani Baranauskas
  Instituto de Computação – UNICAMP
- Prof. Dr. Marcos Lordello Chaim
  Empresa Basileira de Pesquisa Agropecuária - EMBRAPA

FireWeb: Uma Ferramenta de Suporte aos Testes de Regressão de Aplicações Web

Campinas, 09 de dezembro de 2003.

Profa. Dra. Eliane Martins (Orientadora)

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito parcial para obtenção do título de Mestre em Computação na Área de Engenharia de Software.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Fidelis, Wennder Indalécio Oliveira

F448f FireWeb: uma ferramenta de suporte aos testes de regressão de aplicações Web / Wennder Indalécio Oliveira Fidelis -- Campinas, [S.P. :s.n.], 2003.

Orientadora : Eliane Martins

Trabalho final (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Computação.

1. Software - Testes. 2. Engenharia de software. 3. Sites da web. I. Martins, Eliane. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

RESUMO

A rápida difusão da Internet e novas tecnologias estão produzindo um significativo crescimento da demanda de aplicações Web, sendo que cada vez mais são exigidos requisitos de usabilidade, confiança, funcionalidade e segurança.

Esse trabalho propõe uma estratégia de testes de regressão voltada para aplicações Web que visa auxiliar os testadores de aplicações Web na tarefa de selecionar dentre os testes aplicados durante o desenvolvimento, aqueles que deverão ser reexecutados durante a manutenção, levando em consideração as modificações realizadas na aplicação. A estratégia proposta é resultado da análise das estratégias hoje existentes para testes de regressão, buscando identificar dentre as mesmas, aquela cujos princípios mais se adequassem ao contexto de manutenção de aplicações Web.

Também é apresentada a ferramenta "FireWeb", a qual automatiza muitos dos passos da estratégia proposta, tornando o processo mais rápido e barato.

ABSTRACT

The fast Internet diffusion and the new technologies are producing a significant growth in the Web applications demand, whereas more and more requisites such as user-friendly interface, dependability, functioning and security are being required.

This work suggests the strategy of using regression tests made for Web applications, which in this case, are meant to help the Web applications testers in the task of selecting among the applied tests during the developing phase, those which will have to be re-run during the maintenance, taking into consideration the changes made in the application. The strategy presented is the result of the analysis of the existing regression tests, with the purpose of identifying among them, those that principles fit more adequately to Web applications maintenance context.

The FireWeb tool is also presented; it makes several steps of the presented strategy automatic making the process faster and cheaper.

Dedico esse trabalho, especialmente aos meus pais, Hercília Maria de Oliveira e Wilson Fidelis, e ä minha namorada Fabíola (futura esposa), sem os quais o presente trabalho não teria sido realizado.

AGRADECIMENTOS

A Deus, por mais essa conquista.

À professora Eliane, pelos seus ensinamentos durante as aulas de “Verificação e Validação” no curso de “Especialização em Engenharia de Software” do Instituto de Computação”, as quais me fizeram despertar o interesse pela área de testes de software; e também pela sua dedicação durante a orientação do presente trabalho, sem a qual o mesmo não seria possível.

Aos demais professores do Instituto de Computação, em especial, à professora Thelma Chiossi, por ter acreditado em mim, concedendo-me uma carta de recomendação para o processo seletivo de admissão ao mestrado.

Aos colegas do Instituto de computação, em especial aqueles que se tornaram meus amigos: Dinailton, Adão, Josko, Ganhoto, Luis e Marcos Valério.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1- INTRODUÇÃO

1.1 Contexto

A rápida difusão da Internet e novas tecnologias estão produzindo um significativo crescimento da demanda de aplicações Web, sendo que cada vez mais são exigidos requisitos de usabilidade, confiança, interoperabilidade e segurança [GIU02].

Além disso, a relevância econômica das aplicações Web aumenta a importância de controlar e melhorar sua qualidade. Como conseqüência, uma alta demanda está imergindo para metodologias e ferramentas para garantia da qualidade de sistemas baseados em Web [FIL01].

Um fator importante a ser considerado é o fato de que a manutenção é a fase do ciclo de vida de um sistema, seja ele Web ou não, responsável por consumir a maior parte de todo o esforço, cerca de 70% [VOL01]. Estudos mais recentes mostram que esse custo pode chegar a 90% do custo total, dependendo da aplicação [PIG97].

Por isso, na manutenção dos sistemas, para que continue sendo garantida a qualidade do software, encontram-se as atividades de teste de regressão, as quais visam minimizar ao máximo o número de falhas incorporadas ao software após as modificações sofridas na manutenção.

Associados aos testes, um problema em nossas empresas de Tecnologia da Informação é o fato de que poucas possuem especialistas da área de testes, ferramentas e ambientes de testes adequados para tratar de testes em grande escala. A maioria das manutenções dos aplicativos é testada superficialmente. Em geral, são retirados alguns dados de produção para se fazerem os testes [VOL01].

Pressman em [PRE02] cita que sempre que uma modificação é realizada em um software, “efeitos colaterais” podem ser gerados, e quando os mesmos são responsáveis pelo aparecimento de falhas no software, às vezes ouvimos alguém dizer: “... mas tudo que eu fiz foi mudar essa instrução”. É para evitar que frases como essa

sejam ditas e ouvidas, é que existem os Testes de Regressão, ou seja, para garantir que aquilo que funcionava antes da manutenção continue funcionando após a mesma.

Quando tratamos especificamente de testes de aplicações Web, nos situamos num campo ainda muito imaturo, em que há uma grande falta de metodologias que amparem os testes desse tipo de aplicação. Até o presente, muitos dos aspectos de testes ainda não foram suficientemente investigados; e muitas questões ainda necessitam de ser estudadas. Como exemplo, em virtude da grande heterogeneidade de tecnologias utilizadas para implementar tais aplicações, a tarefa de definir e validar modelos de testes que representem os componentes de software em um nível correto de granularidade e os seus relacionamentos ainda é uma questão aberta que necessita ser estudada [GIU02].

Como conseqüência dessa imaturidade, várias questões de testes de software, como a definição de critérios de cobertura e estratégias para conduzir os testes de integração, ou a aplicação de técnicas que permitam a automação de algumas atividades dos testes, ainda necessitam também ser estudadas [GIU02].

1.2 Motivação

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

- devido à rápida difusão da Internet, as aplicações Web hoje são uma tendência de mercado e necessitam possuir um nível alto de qualidade;
- importância dos testes durante a manutenção de sistemas, para garantir que as modificações não inseriram falhas em partes não modificadas;
- falta de metodologias que amparem o teste de aplicações web, sobretudo durante a manutenção;

1.3 Objetivos

Esse trabalho propõe uma estratégia de testes de regressão voltada para aplicações Web, a qual visa auxiliar os testadores de aplicações Web na tarefa de

selecionar dentre os testes aplicados durante o desenvolvimento, aqueles que deverão ser reexecutados durante a manutenção, levando em consideração as modificações realizadas na aplicação.

A estratégia proposta é fruto da análise das estratégias hoje existentes para testes de regressão, buscando identificar aquela cujos princípios mais se adaptassem ao contexto de manutenção de aplicações Web.

O presente trabalho também apresenta a ferramenta "*FireWeb*", a qual automatiza muitos dos passos da estratégia proposta. O objetivo é demonstrar a capacidade de automação da estratégia, e com isso, tornar o processo de validar aplicações Web mais rápido e menos oneroso.

Além disso, o presente trabalho espera contribuir para a disseminação dos conceitos de testes e testes de regressão e, com isso ajudar a sensibilizar a comunidade de desenvolvedores de software, da importância que os testes têm na garantia de qualidade.

1.4 Organização

O presente material está organizado da seguinte forma:

No capítulo 2, "Aplicações Web – Uma Breve Introdução", é apresentada uma visão geral das aplicações Web, dando ênfase às suas arquiteturas.

No capítulo 3, "Revisão de Testes de Software", são apresentados os conceitos básicos de testes de software, tais como, tipos de testes, suas fases e automação dos mesmos. Nesse capítulo ainda são discutidas algumas diferenças entre os testes de softwares tradicionais em relação aos testes de aplicações Web.

No capítulo 4, "Testes de Regressão", são discutidos os aspectos inerentes às tarefas de testar um software após a ocorrência de modificações. São apresentadas as limitações e as situações em que os testes de regressão devem ser executados, bem como as técnicas existentes para apoiar esse tipo de teste.

No capítulo 5, "Estratégia Proposta", são apresentados os objetivos da estratégia proposta no presente trabalho, suas particularidades, procedimentos de execução e sua

avaliação.

No capítulo 6, "*FireWeb* – *Firewall* para Web", é apresentada a ferramenta construída para demonstrar a capacidade de automação da estratégia proposta. São demonstrados seus objetivos, funcionalidades e limitações.

No capítulo 7, "Conclusão e Trabalhos Futuros", são apresentadas as conclusões do presente trabalho, sua contribuição para a área de testes de software, bem como os aspectos que ainda necessitam de exploração em trabalhos futuros.

No apêndice A e apresentada uma relação de ferramentas comerciais para testes de aplicações Web. As ferramentas constantes nesse apêndice não tratam da seleção dos testes a serem reexecutados durante os testes de regressão. Daí um motivo para a necessidade de construção da Ferramenta "*FireWeb*" a qual é descrita no capítulo 6 do presente trabalho.

Capítulo 2 - APLICAÇÕES WEB – UMA BREVE INTRODUÇÃO

Nesse capítulo será apresentada uma visão geral das aplicações Web. Serão apresentados os componentes que compõem esse tipo de aplicação, suas arquiteturas principais e os componentes que as compõem.

2.1 Arquitetura de aplicações distribuídas

Em uma arquitetura distribuída, os componentes são agrupados em módulos de serviços relacionados. Arquiteturas distribuídas são usadas tanto para sistemas cliente-servidor, como para sistemas Web [HUN01].

Um sistema distribuído pode ser classificado como: “cliente-magro”, quando a maioria do processamento ocorre do lado servidor e em “cliente-gordo”, quando o oposto ocorre, ou seja, quando a maioria do processamento ocorre do lado cliente. Nas figuras 2.1 e 2.2 seguintes, são mostrados os modelo “cliente-magro” e “cliente-gordo” respectivamente [HUN01].

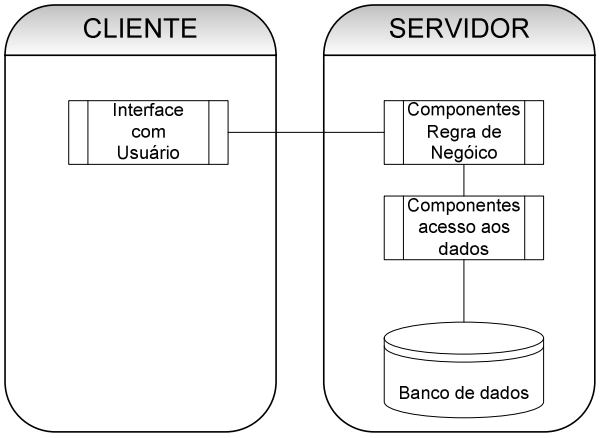


Figura 2.1 Modelo “cliente-magro”

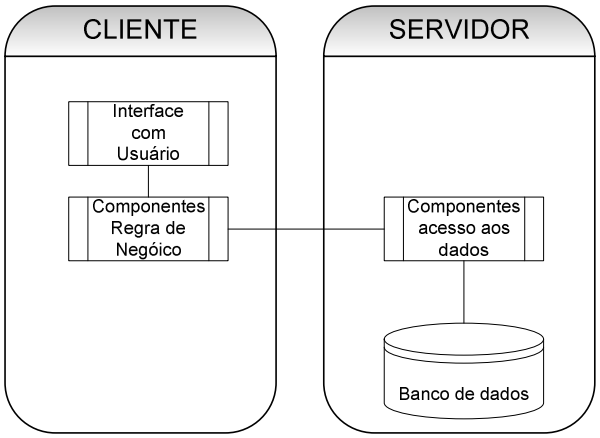


Figura 2.2 Modelo “cliente-gordo”

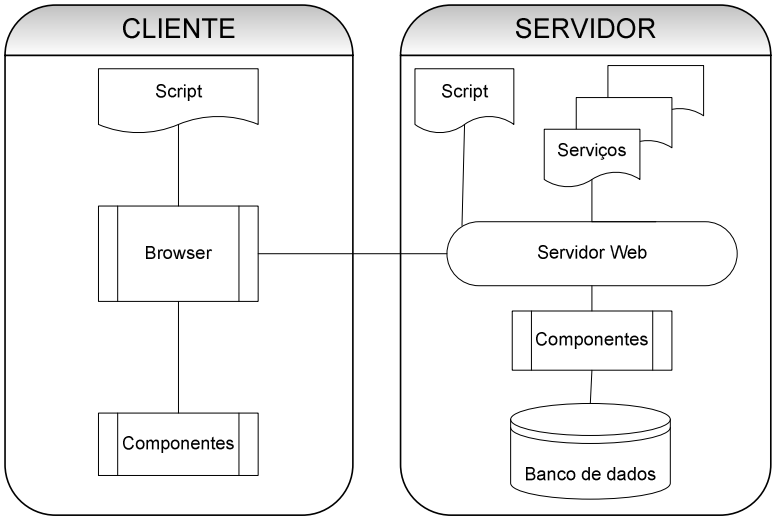


Figura 2.3 Arquitetura Web baseada no modelo “cliente-gordo”

Modelos “cliente-magro”, geralmente, são utilizados quando as máquinas clientes não detêm uma capacidade grande de processamento, como no caso dos PDA (*Personal digital assistants*), os quais devido ao seu tamanho reduzido, possuem ainda pouca capacidade de processamento. Todavia, quando os clientes geralmente são Desk-Tops, os quais possuem uma boa capacidade de processamento, os modelos “cliente-gordo” geralmente são os empregados. Nas figuras 2.3 e 2.4 são exibidos os modelos “cliente-gordo” e “cliente-magro” aplicados aos sistemas Web [HUN01]. Nas seções seguintes, são apresentados dois modelos de aplicações “cliente-servidor”, os

tradicionais sistemas cliente-servidor e as aplicações Web.

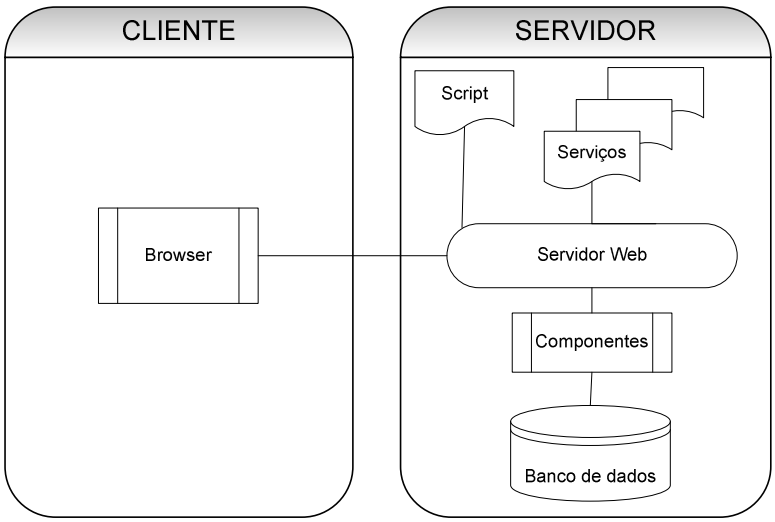


Figura 2.4 Arquitetura Web baseada no modelo “cliente-magro”

2.1.1 Sistemas Cliente-Servidor Tradicionais

Uma aplicação de acesso a dados tipicamente consiste de quatro elementos:

- **interface com o usuário**: as aplicações responsáveis por captar os dados fornecidos pelo usuário e também por exibir os resultados;
- **regras do negócio**: aplica regras, faz cálculos e manipula dados.
- **acesso aos dados**: trata as consultas e as atualizações dos dados no banco de dados e envia os dados de volta ao cliente.
- **armazenamento dos dados**: guarda as informações.

Sistemas cliente-servidor requerem, no mínimo, dois computadores: um fazendo o papel de “servidor” e outro fazendo o papel de “cliente”. O “servidor” recebe os dados das requisições vindas do “cliente”, manipula-os através de procedimentos que representam as “regras do negócio” e, então, devolve os resultados ao “cliente”.

2.1.2 Aplicações Web Cliente-Servidor

Uma aplicação Web cliente-servidor tipicamente agrupa seus componentes em três camadas:

- componentes de interface com o usuário;

- componentes de regras do negócio;
- componentes de serviços de acesso a dados.

Desses, apenas os de interface com o usuário ficam do lado cliente da aplicação, os outros ficam do lado servidor.

Aspectos como: processamento, performance, escalabilidade e manutenção do sistema dependem de como foram projetados e implementados tais componentes. As aplicações Web possuem algumas diferenças das aplicações tradicionais. Algumas dessas diferenças são discutidas na seção 2.2, abaixo.

2.2 Aplicações Web X Aplicações Tradicionais

Ter um conhecimento de como as aplicações Web funcionam, suas diferenças em relação aos outros tipos de plataformas, podem auxiliar os testes das mesmas. As diferenças das aplicações Web em relação às aplicações tradicionais vão desde o hardware até o software. Por exemplo, em um sistema mainframe, geralmente o que temos é um computador central (mainframe) e vários “terminais burros” ligados a ele. Todo o processamento é realizado no mainframe. A única coisa que os “terminais burros” fazem é receber os dados de entrada, os quais são repassados ao mainframe, e, depois, emitir os resultados, caso estes não sejam enviados diretamente às impressoras.

Sistemas baseados em máquinas Desktop encarregam-se de fazer todo o serviço (tanto o de cliente, quanto o de servidor). Nesse caso como tudo é processado em um único computador, não se faz necessário o uso de rede de comunicação.

Os sistemas Web enquadram-se como um tipo de sistema cliente-servidor. Uma das grandes diferenças em relação aos tradicionais sistemas cliente-servidor é o fato de o cliente, no caso dos sistemas Web, ser um browser [HUN01].

Para que os testes de aplicações Web possam ser bem guiados, é importante e necessário para os testadores, um mínimo de conhecimento sobre os componentes que fazem parte desse tipo de aplicação. Na seção 2.3 abaixo, são apresentados, de forma breve, esses componentes.

2.3 Componentes de uma aplicação Web

Ter um conhecimento dos componentes internos de uma aplicação Web, e de como esses componentes interagem uns com os outros, contribui para testar melhor o sistema. Isso faz com que o testador possa testá-lo a partir da perspectiva do desenvolvedor, o que pode ser útil na hora de determinar a estratégia de testes e de identificar a causa dos erros [HUN01].

Devem ser analisados os componentes de hardware e software, desde o lado cliente (como browsers, plug-ins e objetos embutidos) até os componentes do lado servidor (como componentes da aplicação servidora, aplicações de bancos de dados e outras). Tal conhecimento ajuda a responder os questionamentos abaixo listados, os quais podem ocorrer durante os testes de um sistema Web [HUN01]:

- Quais tipos de plug-ins são usados pela aplicação em teste? Quais são as implicações nos testes associados a esses plug-ins?
- Como a distribuição dos componentes do lado servidor afeta o projeto e a estratégia de testes?
- Quais servidores Web e de Bancos de Dados são suportados pela aplicação? Como a conexão entre a Web e o Banco de Dados está implementada, e quais são as implicações de testes associados?
- Como os testes podem ser particionados para consertar os componentes problemáticos?

Um sistema Web é composto de componentes de hardware, de software e de usuários. Aqui, o estudo enfocará os componentes de software.

Aplicações Web exploram facilidades de navegação e de interação de páginas HTML para prover e questionar informações do/para o usuário [FIL01].

Um meta-modelo representado em UML (*Unified Modeling Language*) é proposto para retratar a estrutura genérica de aplicação Web, conforme é exibido na figura 2.5 [FIL01].

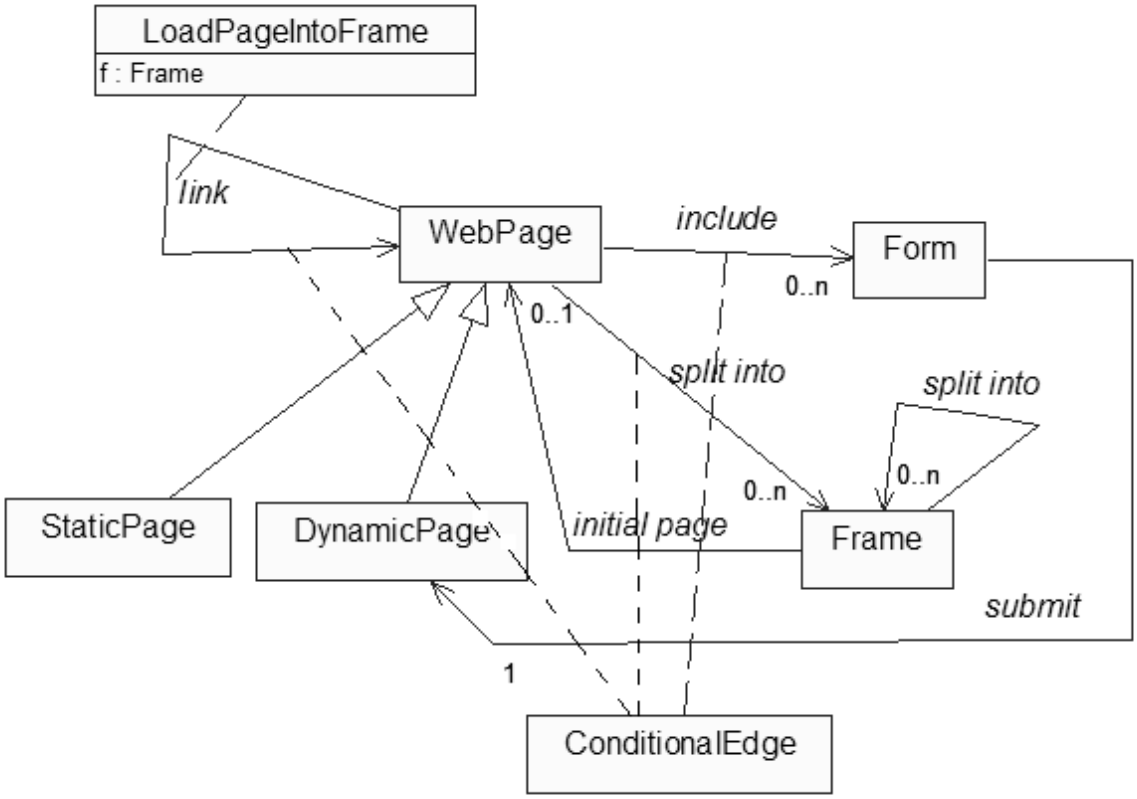


Figura 2.5 Modelo genérico de uma aplicação Web [FIL01]

Analisando o modelo da figura 2.5, vemos que no centro do modelo aparece a classe "*Web page*" (página Web). Uma "*Web page*" contém a informação que é exibida ao usuário e os "*links"* de navegação que levam as outras "*Web pages*". Ela também inclui facilidades de organização e de interação como: "*frames*" e "*forms*". A navegação de uma página a outra página é representada pela auto-associação da classe "*Web page*" nomeada de "*link*". Pelo modelo, vemos que uma página pode ser de dois tipos: estática ou dinâmica. Enquanto o conteúdo de uma página *estática* é fixo, o conteúdo de uma página *dinâmica* é gerado em tempo de execução pelo servidor. Um "*frame*" é uma área retangular disposta dentro da página, onde a navegação pode acontecer independentemente das outras áreas da "*Web page*". A divisão da página em "*frames"* pode ser recursiva (auto-associação "*split-into*" na classe "*frame*").

Quando a navegação em uma página força a "carga" de outra página em um "*frame"* diferente do atual, o "*frame*" alvo torna-se um parâmetro da classe de

associação "*LoadPageIntoFrame".* Em HTML, entradas/requisições de usuários podem ser tratadas pelo uso de "*forms*" (formulários). Uma página Web pode incluir um número qualquer de "*forms"* (associação *include*).

Cada *form* é caracterizado pelas variáveis de entrada que são fornecidas pelos usuários. Valores coletados pelos "*forms"* são submetidos ao servidor Web através de um link especial "*submit*". Como "*links"*, "*frames"* e "*forms"* são parte do conteúdo da página Web, e, para páginas dinâmicas o conteúdo pode depender das variáveis de entrada, até a organização da página, em geral, é não-fixa e depende dos valores de entrada. Daí, a razão para a classe de associação *"ConditionalEdge"*, a qual opcionalmente adiciona uma condição booleana, função da entrada das variáveis, representando a existência de uma condição da associação que pode ser um "*link*", "*include*" ou "*split-into*". O alvo página, "*form"* ou "*frame"* é referenciado pelo código da página dinâmica apenas quando os valores de entrada satisfazem a condição em "*ConditionalEdge*" [FIL01].

2.3.1 Componentes do Lado Servidor

Qualquer computador que provê serviços a outro computador é um "servidor". Um único computador pode hospedar vários servidores (softwares). No que tange a hardware, as qualidades necessárias a um servidor, em geral, são favoráveis a todos os computadores, como: alta performance, alta taxa de transmissão de dados, escalabilidade e confiabilidade [HUN01].

Quanto aos sistemas operacionais dos servidores, os mesmo devem ser mais robustos que os sistemas operacionais de estações de trabalho. Windows 95 e 98, por exemplo, não oferecem a confiabilidade ou performance requerida pela maioria dos servidores. Já sistemas operacionais, como Unix e Windows NT, oferecem fortes características de segurança e ferramentas de administração, além da escalabilidade e confiabilidade requeridas pelos servidores [HUN01].

- **Servidores Web ou servidores http**: armazenam páginas Web ou arquivos HTML e seus conteúdos associados. Servidores Web tornam seu conteúdo disponível aos componentes clientes. Servidor Web é o principal tipo de servidor para aplicações Web [HUN01].

- **Servidores de Bancos de Dados**: atuam como repositórios de dados para as aplicações Web. A maioria das aplicações Web utiliza bancos de dados relacionais. Fornecedores como de SGBD (Sistemas Gerenciadores de Bancos de Dados) são: Microsoft, Oracle e Sybase [HUN01].
- **Servidores de Aplicação:** esse termo é utilizado para referenciar um conjunto de componentes que estendem seus serviços a outros componentes [HUN01].

2.3.2 Componentes do Lado Cliente

O lado cliente de um sistema Web freqüentemente é formado por uma variedade de elementos de hardware e software. Muitos produtos podem estar presentes em um único sistema. A natureza heterogênea de hardware, elementos de rede, sistemas operacionais e software no lado cliente podem tornar o teste de tais aplicações um desafio. Abaixo, são discutidos os principais componentes clientes de uma aplicação Web, que são: Web Browsers e componentes Add-on/Plug-in [HUN01]:

- **Web Browsers**: são aplicações que recebem, montam e exibem as páginas web. No modelo cliente-servidor de um sistema Web, os browsers sãos os clientes. Os browsers requisitam as páginas Web aos servidores Web. Os servidores Web, então, localizam as páginas requisitadas e as encaminham de volta aos browsers. As informações são exibidas aos usuários via páginas Web pela interpretação das instruções de código em HTML, que são enviadas aos browsers pelo servidor Web. Quando uma página contém arquivos multimídia, tais arquivos estão armazenados em locais independentes, sendo que na página Web existem apenas links que apontam para o endereço desses arquivos.
- **Componentes Add-on/Plug-in**: são softwares adicionais que ficam do lado do cliente para dar suporte a várias formas de interatividade e animação dentro das páginas Web. Java applets e controle ActiveX são exemplos desse tipo de componente.

Em se tratando de aplicações comerciais, um outro aspecto importante a ser

levado em consideração na hora de se testar aplicações Web, é a forma de conexão da aplicação com o banco de dados. A seção 2.4 abaixo apresenta os principais meios de comunicação Web X banco de dados atualmente.

2.4 Conexão Web a banco de dados

A comunicação entre usuários, servidores Web e servidores de bancos de dados é facilitada por certas extensões e modelos de scripts.

No lado servidor, os dados residem num banco de dados. No cliente, o usuário é representado por requisições enviadas ao servidor Web. Sendo assim, prover a conexão entre requisições do servidor Web e o banco de dados é a função chave das aplicações Web. Existem várias formas de estabelecer essa conexão, as mais comuns são [HUN01]:

- **Common Gateway Interface (CGI)**: o CGI é um protocolo de comunicação que o servidor Web usa para comunicar-se com outras aplicações. Scripts CGI permitem aos servidores Web acessarem bancos de dados, entre outras coisas. Aplicações CGI usualmente são escritas em "*Practical Extraction and Report Language*" (perl), embora elas possam ser escritas em outras linguagens de programação, tais como: C, C++ e Visual Basic. O uso de scripts CGI não é muito interessante pelo fato dos mesmos serem arquivos executáveis separados. Caso haja um volume grande de invocações de aplicações CGI, estas estarão consumindo muito recurso do servidor Web.
- **Web Server Extension-Based Programs**: geralmente são aplicações no formato de DLL. Essas DLL expõem seus diferentes tipos de funcionalidades, incluindo conexão a bancos de dados. Elas, por executarem no mesmo espaço de memória que o software do servidor Web, são melhores em termos de uso de recursos do que os CGI, que utilizam arquivos executáveis separados. A principal desvantagem de "Web Server Extension-Based Programs", é que, por serem escritas em linguagens como C, C++ e Visual Basic, as mesmas são binárias. Qualquer mudança no código das mesmas requer uma recompilação. Além disso, essas linguagens são de mais difícil

aprendizagem, o que irá requerer profissionais mais bem treinados. As linguagens de scripts por serem mais fáceis destacam-se nesse aspecto.

2.5 Considerações Finais

No presente capítulo foram discutidos os aspectos gerais das aplicações Web.Nele pudemos analisar que as aplicações Web são aplicações distribuídas e caracterizam-se como aplicações “cliente-servidor”.

Foram apresentados os principais modelos de aplicações distribuídas, que são os modelos “cliente-magro”, onde a maior parte do processamento ocorre do lado servidor e “cliente-gordo”, onde o cliente também é encarregado de efetuar boa parte do processamento, evitando assim a sobrecarga de processamento no lado servidor da aplicação.

Analisando uma aplicação cliente-servidora tradicional que faz acesso a dados, vimos que ela geralmente consiste de quatro elementos que são:

- **interface com o usuário**: as aplicações responsáveis por captar os dados fornecidos pelo usuário e também por exibir os resultados;
- **regras do negócio**: aplica regras, faz cálculos e manipula dados.
- **acesso aos dados**: trata as consultas e as atualizações dos dados no banco de dados e envia os dados de volta ao cliente.
- **Armazenamento dos dados**: guarda as informações.

Uma aplicação Web diferencia-se de uma aplicação cliente-servidora tradicional principalmente pelo fato de o cliente ser um browser [HUN01]. Os browsers requisitam as páginas Web aos servidores Web, os quais por sua vez, localizam as páginas requisitadas e as encaminham de volta aos browsers.

As aplicações Web são formadas por uma série de componentes que residem do lado servidor ou do lado cliente. Dentre os componentes do lado servidor, os principais são [HUN01]:

- **Servidores Web**: armazenam as páginas Web ou arquivos HTML e seus conteúdos associados;

- **Servidores de Bancos de Dados**: atuam como repositórios de dados para as aplicações Web.
- **Servidores de Aplicação**: conjunto de componentes que estendem seus serviços a outros componentes.

No cliente os principais componentes são:

- **Web browsers**: são aplicações que recebem, montam e exibem as páginas Web. As informações são exibidas aos usuários via páginas Web pela interpretação das instruções de código em HTML, que são enviadas aos browsers pelo servidor Web.
- **Componentes Add-on/Plug-in**: são softwares adicionais que ficam do lado cliente para dar suporte a várias formas de interatividade e animação dentro das páginas Web.

Portanto, para que os testes de aplicações Web possam ser mais bem conduzidos, é importante e necessário para os testadores, um mínimo de conhecimento sobre o funcionamento dessas aplicações.

Capítulo 3 - REVISÃO DE TESTES DE SOFTWARE

No mundo moderno, o software passou a exercer, de forma intrínseca, um importante papel como apoio aos negócios das empresas. Essa importância tende a crescer em um mundo que as atividades e os produtos tendem a depender cada vez mais dessa ferramenta.

Por outro lado, desde os primeiros computadores comerciais, os softwares utilizados têm se caracterizado, em sua maioria, por apresentarem um grande número de defeitos que afetam a sua usabilidade, funcionalidade, segurança e confiabilidade, gerando um impacto decisivo nos negócios, o que resulta grandes prejuízos aos seus consumidores.

Em um mundo globalizado, em que a Internet é um importante apoio aos negócios, um teste de software mal feito pode significar um caminho aberto para diversos problemas, como por exemplo: fraudes, incorreções e bloqueios do “site”. Muitas vezes, os usuários acabam abandonando o “site” insatisfeitos com a sua usabilidade e/ou funcionalidade, causando enormes prejuízos financeiros e para a imagem da organização [RIO03].

A crescente visibilidade do software como um elemento do sistema e os “custos” de atendimento associados com uma falha são forças motivadoras para um teste rigoroso e bem planejado. É comum, uma organização de desenvolvimento de software gastar entre 30% e 40% do esforço total do projeto com testes. A rigor, o teste de software que envolve vidas (p. ex., controle de vôo, monitoramento de reatores nucleares) pode custar de três a cinco vezes mais do que todos os outros passos de engenharia de software combinados [PRE02].

3.1 Objetivos

Teste é uma forma de verificação dinâmica que consiste em executar o programa

com um conjunto de dados de entrada e determinar se ele se comporta conforme o esperado, isto é, de acordo com sua especificação. Em geral, é impossível testar um programa exaustivamente; o importante é selecionar um conjunto finito de casos de testes que permitam testá-lo adequadamente [MAR00].

Glen Myers [MYE79] enumera algumas regras que podem servir como objetivos do teste:

- teste é um processo de execução de um programa com a finalidade de encontrar um erro.
- um bom caso de testes é aquele que tem alta probabilidade de encontrar um erro ainda não descoberto.
- um teste bem-sucedido é aquele que descobre um erro ainda não descoberto.

Os conceitos de erro, de falha e de defeito, os quais serão tratados mais adiante, seguem o IEEE Std. Glossary of Software Engineering Terminology, padrão 610.12/1990, onde os mesmos são definidos como:

- **erro**: engano cometido por um desenvolvedor (analista, projetista, programador);
- **falha**: manifestação do erro (uma especificação ou código incorretos);
- **defeito**: evento notável ao usuário, ativação da falha.

Quanto à completeza dos testes, Pressman [PRE02] afirma que “teste completo não é possível”. A quantidade de permutações de caminhos, mesmo para um programa de tamanho moderado, é excepcionalmente grande. Por essa razão, é impossível executar todas as combinações de caminhos durante o teste.

3.2 A importância dos testes

Em [RIO03] é apresentado o que já foi dito por Bohem (1979), ou seja, quanto mais tarde um erro for identificado, mais caro fica para corrigi-lo. E mais ainda, os custos de se descobrir e de se corrigir erros no software aumentam exponencialmente

na proporção em que o trabalho evolui através das fases do projeto de desenvolvimento.

Além disso, deve ser ressaltada a importância dos testes na manutenção de sistemas [RIO03], fase esta que, segundo estudos apresentados em Pigoski [PIG97], a partir dos anos 90 já corresponde a 90% do custo total com o sistema.

Na fase de manutenção, devem ser aplicados os *Testes de Regressão* para garantir que as modificações inseridas não provocaram erros em partes não modificadas [RIO03].

3.3 Fases do teste

Existem diversas fases de teste, cada fase devendo ser preparada ao longo do desenvolvimento. Em geral, o que se sugere é um processo de desenvolvimento em "V", mostrado na figura 3.1, englobando tanto as fases de desenvolvimento, quanto às fases de testes [MAR00].

O processo de testes inicia-se com os ***testes de unidade***, que visam verificar se cada módulo ou unidade satisfaz à sua especificação, estabelecida no Projeto Detalhado. Após testar separadamente cada módulo, estes são agrupados para compor os subsistemas, conforme a arquitetura do sistema definida no Projeto Preliminar, sendo essa fase de ***testes de integração***. O objetivo dos testes de integração é encontrar falhas de interfaceamento entre os módulos e os subsistemas. Os ***testes de validação*** visam determinar se o software satisfaz aos requisitos especificados na fase de análise. E, finalmente, **os *testes de sistema*** visam exercitar o sistema como um todo, incorporando todos os componentes (hardware e software) para determinar se o sistema completo satisfaz à sua especificação [MAR00].

3.4 Técnicas e critérios de testes

Em [PRE02] vemos que os testes de software estão divididos em duas subclasses maiores, que são:

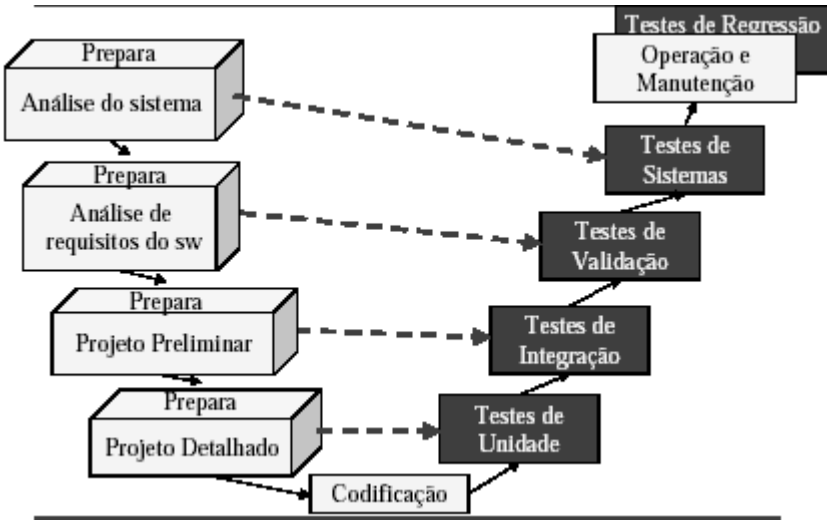


Figura 3.1 Processo de desenvolvimento em "V" [MAR00]

- **testes baseados na implementação (caixa branca)**: baseados na implementação, a partir de informações do código do programa [MAR00].
- **testes baseados na especificação (caixa preta)**: também chamados de testes comportamentais, focalizam os requisitos funcionais do software [PRE02]. Não se preocupam com o código interno do programa [RIO03].

3.5 Testes baseados na implementação (caixa branca)

Em testes baseados na implementação, a seleção dos testes é baseada na informação obtida a partir do código do programa. Assim sendo, esses métodos são também chamados de testes caixa branca ou caixa de vidro ou ainda teste estrutural, significando que a estrutura da implementação é considerada para a escolha dos testes.

Os testes caixa branca podem ser classificados segundo dois pontos de vista ortogonais: busca de falhas ou estrutura do programa.

Do ponto de vista de busca de falhas, considera-se que, para que as falhas sejam reveladas, é necessário que elas levem a ocorrência de defeitos, e isso só acontece: (i) se as falhas existentes são ativadas, isto é, o local onde elas se encontram

no programa é executado; (ii) se o estado do programa é infectado pela ativação das falhas, criando erros; e (iii) se os erros são propagados pelo resto do programa, infectando outros estados até causar um defeito.

Em relação à estrutura do programa, considera-se que o mesmo contém operações (operadores, comandos) e operandos (variáveis e itens de dados), cada qual podendo conter falhas. A revelação de uma falha depende de vários fatores: (i) do componente estrutural (operação ou operando) em que ela se encontra; (ii) da execução de uma determinada seqüência de comandos; ou ainda, (iii) dos valores das variáveis em um determinado momento da execução.

Baseando-se nesses fatores, os critérios de testes (métodos de seleção) serão classificados em:

- **baseados na estrutura**: visam exercitar os diferentes componentes estruturais do programa;
- **baseados no fluxo**: visam exercitar as diferentes seqüências de execução de comandos ou de definição-uso de dados;
- **baseados no estado**: visam exercitar os diferentes estados do programa. O estado é determinado pelos valores de suas variáveis em um momento específico de execução.

3.5.1 Grafos de fluxo de controle

No grafo de fluxo de controle, cada círculo representa um segmento de código que é executado seqüencialmente, possivelmente terminando com um comando de desvio. Cada ramo (arco) representa a transferência de controle (desvio) entre os segmentos [MYE79].

As figuras 3.2 e 3.3, respectivamente, exibem como exemplo o procedimento “média” com a identificação dos seus “nós” e o seu grafo de fluxo de controle.

Nó	PROCEDIMENTO média; Este procedimento calcula a média de 100 ou menos números situados entre valores limites; calcula também a soma e o total de números válidos. INTERFACE RETORNA média total.entradas, total.válidas; INTERFACE ACEITA valor, mínimo, máximo; TIPO valor[1:100] É VETOR DE ESCALAR; TIPO média, total.entradas, total.válidas; mínimo, máximo, soma É ESCALAR; TIPO I É INTEIRO
1	I = 1;
1	total.entradas = total.validas = 0;
1	soma = 0;
1	FAÇA ENQUANTO
2	valor[i] <> -999
3	E total.entradas < 100
4	incremente total.entradas de 1;
5	SE valor[i] >= mínimo
6	E valor[i] <= máximo
7	ENTÃO incremente total.validas de 1;
7	soma = soma + valor[i]
7	SENÃO pule
8	FIM-SE
8	incremente i de 1;
9	FIM ENQUANTO
10	SE total.válidas > 0
11	ENTÃO média = soma / total.válidas;
12	SENÃO média = 999;
13	FIM-SE
	FIM-MEDIA

Figura 3.2 Procedimento “média” com a identificação dos nós

3.5.2 Critérios baseados na estrutura

Esses critérios consideram apenas a cobertura dos componentes estruturais do programa, sem levar em conta o seu contexto de execução, sendo classificados como testes de ativação de falhas. Alguns deles são listados abaixo [MAR00]:

- **critérios baseados nas instruções**: requer que cada instrução do programa seja executada ao menos uma vez;

- **critérios baseados nas decisões**: requer que cada ramo de uma decisão (correspondendo a uma saída “V” e uma saída “F”) seja executado pela menos uma vez. Esse critério também é conhecido como *teste de ramos*;
- **critérios baseados nos dados**: requer que cada item de dado seja exercitado pelo menos uma vez durante os testes, onde um item de dado pode ser uma variável escalar, um campo de uma variável estruturada (como o tipo **record** em Pascal, ou **struct** em C, por exemplo), ou um elemento de uma variável indexada.

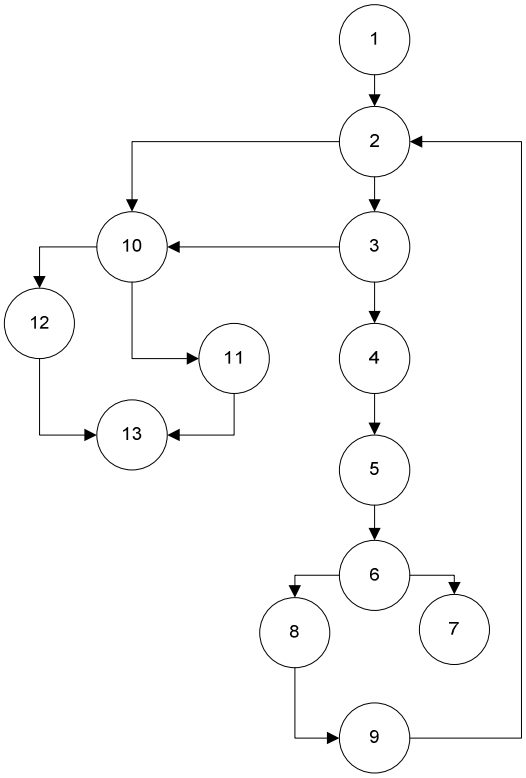


Figura 3.3 Grafo de fluxo de controle para o procedimento média

3.5.3 Critérios baseados no estado

Esses testes consideram os valores que podem ter as expressões contidas no programa, incluindo valores de sub-expressões e de variáveis. Esses critérios têm por objetivo fazer com que, nos locais onde eventualmente existam falhas, o estado dos dados seja infectado com valores incorretos [MAR00].

3.5.4 Critérios baseados no fluxo

Esses critérios consideram os componentes estruturais do programa no seu contexto de execução, ou seja, eles consideram seqüências de comandos ou de definição/uso de dados. Eles objetivam a propagação do estado errôneo: caminhos e execução são escolhidos com base na sua capacidade para a propagação de estados infectados pela ativação de falhas [MAR00].

Dividem-se em:

- **critérios baseados no fluxo de controle**: consideram a cobertura dos caminhos do grafo de fluxo.
- **critérios baseados no fluxo de dados**: utilizam as relações entre os pontos do programa onde as variáveis são definidas e o ponto onde elas são usadas como base para a geração dos testes.

3.6 Testes baseados na especificação (caixa preta)

Os testes caixa preta visam verificar a funcionalidade e a aderência aos requisitos em uma ótica externa ou do usuário, sem se basear em qualquer conhecimento do código e da lógica interna do componente testado [RIO03].

Como os testes caixa preta baseiam-se nas especificações, uma especificação de boa qualidade que seja testável faz-se necessária. Para isso, a especificação deve ser não ambígua (cada termo tem uma e somente uma definição), consistente (cada termo é usado somente de uma forma) e completa (todas as informações necessárias e suficientes para os testes foram definidas). Com isso, fica mais fácil identificar com precisão os dados de entrada e os resultados esperados [MAR00].

Os testes caixa preta tentam encontrar erros das seguintes categorias [PRE02]:

- funções incorretas ou omitidas;
- erros de interface;
- erros de estrutura de dados ou de acesso à base de dados externa;
- erros de comportamento ou desempenho;

- erros de iniciação e término.

Com a utilização dos testes caixa preta, é possível derivar um conjunto de casos de testes que satisfaça os seguintes critérios [MYE79]:

- casos de testes que reduzem, de um valor que é maior que 1, o número de adicional de testes que precisam ser projetados para atingir um teste razoável;
- casos de testes que nos dizem algo sobre a presença ou ausência de classes de erros, ao invés de um erro associado somente com o teste específico em mãos.

3.7 Testes web x testes tradicionais

Várias metodologias para desenvolvimento de aplicações Web têm surgido tanto na indústria como também nas universidades. Entretanto, a maioria preocupa-se apenas com aspectos externos dessas aplicações, tais como: usabilidade e acessibilidade, deixando de lado aspectos internos como manutenibilidade e testabilidade. Um outro aspecto importante é que, devido à imaturidade no campo de V&V (Verificação e Validação) para aplicações Web, as mesmas encontram-se com uma comum falta de processos para garantir suas qualidades [GIU02].

Até o presente, muitos dos aspectos sistemáticos de testes não têm sido suficientemente investigados, e muitas questões ainda necessitam de ser tratadas. Como exemplo disso, dadas às varias e heterogêneas tecnologias atualmente usadas para implementar aplicações Web, em que, por exemplo, uma única página ASP (*Active Server Pages*) chega a possuir trechos de código em três linguagens de programação distintas, *Visual Basic Script*, HTML e Java *Script*. Por isso, modelos de testes que representem seus componentes em um grau correto de granularidade, ainda são uma questão aberta. Com isso, níveis de testes de aplicações Web, os quais especificariam as diferentes coleções de componentes a serem testados nos níveis de: unidade, integração e sistema, não podem ser precisamente identificados, dada a falta de uma definição bem aceita da representação de unidades de uma aplicação Web serem

testadas [GIU02].

Um dos grandes desafios do teste Web é o fato da diversidade dos ambientes onde os sistemas irão executar. Diferentemente de um sistema mainframe, onde os sistemas geralmente executam em um único sistema operacional, e as aplicações são vendidas e suportadas por um único fornecedor, na maioria das vezes. Um sistema Web consiste de muitos clientes, os quais não se pode ter controle sobre eles, ou seja, a configuração das máquinas clientes não está sob o controle da equipe de desenvolvedores do sistema. No lado servidor, a coisa também não é homogênea, pois o servidor pode suportar uma mistura de hardware e de software e, portanto, pode ser mais complexo do que um mainframe do ponto de vista de testes de configuração e de compatibilidade, os quais validam a capacidade do software de executar em um particular ambiente de "hardware/software/sistema operacional/rede", etc [HUN01] [RIOS03].

Os sistemas Web, como alguns sistemas baseados em Desktop PC, são baseados em GUI (Graphical User Interface) e, portanto, são dirigidos por eventos (pressionar um botão, ativar um menu suspenso, etc.), o que torna também o teste Web mais complexo do que o teste de um sistema batch (processamento em lote) de um mainframe [HUN01].

3.8 Testes web - automação

Existe no mercado uma variedade de ferramentas disponíveis para automatizar os testes Web.

No apêndice "A" deste documento, são exibidas algumas ferramentas estando agrupadas por tipo. Nele são exibidos: seus nomes, uma breve descrição, os requisitos de hardware e software e respectivas URL [DIA01].

3.9 Considerações Finais

Vimos que os testes são uma atividade fundamental à garantia da qualidade dos softwares, os quais hoje têm um papel importante no apoio aos negócios das

empresas. Foram apresentados os objetivos dos testes, sua importância e fases. Os testes são classificados em “caixa branca” quando focam no código fonte do programa para a escolha dos testes a serem executados; e em “caixa preta” quando focam apenas nas funcionalidades que devem ser atendidas pelo software.

No que se refere aos testes de aplicações Web, vimos que até o presente muitos dos aspectos sistemáticos de testes não têm sido suficientemente investigados e muitas questões ainda necessitam ser tratadas. Um dos aspectos que tornam os testes dessas aplicações um desafio, é a diversidade de tecnologias envolvidas na construção das mesmas. Uma única aplicação pode envolver várias linguagens de programação com isso, modelos de testes que representem seus componentes em um grau correto de granularidade ainda é uma questão aberta.

Capítulo 4 - TESTES DE REGRESSÃO

Do ciclo de vida total de um sistema, estudos apontam que cerca de 90% corresponde à fase de manutenção. Durante essa fase, um software sofre modificações por várias razões como correções de problemas não encontrados durante o desenvolvimento, mudanças de especificação, adaptação a um novo ambiente, etc. Baseado no tipo de mudança, a manutenção de sistemas é classificada nos seguintes tipos [PIG97]:

- **adaptativa**: modificações são realizadas para que o software se adapte a um novo ambiente.
- **corretiva**: modificações são realizadas para corrigir erros.
- **preventiva**: modificações são realizadas para facilitar futuras manutenções.
- **evolutiva**: modificações são realizadas para adicionar novas funcionalidades.

Essas modificações podem fazer com que partes do software, que antes funcionavam de forma impecável, parem de funcionar.
Para evitar que isso aconteça, é que existem os testes de regressão, os quais serão descritos em detalhes nesse capítulo [PRE02].

4.1 Características dos testes de regressão

Testes de Regressão são necessários a cada vez que modificações são feitas no software, e consistem na reexecução de um subconjunto dos casos de testes já aplicados para determinar se as alterações não produziram nenhum efeito indesejado [MAR00].

Uma importante diferença entre testes de regressão e testes durante o desenvolvimento é o fato de que durante os testes de regressão um conjunto de testes pré-existente deve estar disponível para poder ser reutilizado [HAR94] [HAR97].

Uma abordagem para testes de regressão consiste em *retestar tudo*, selecionando todos os casos de testes do conjunto já existente. Essa estratégia pode

consumir tempo e recursos excessivos. Uma abordagem alternativa é o *reteste seletivo*, o qual se baseia na seleção de um subconjunto de testes a partir do já existente, usando esse subconjunto para testar o programa modificado [HAR94] [HAR97].

Os conceitos abaixo são importantes quando abordamos os testes de regressão [BIN99]:

- **linha básica**: versão de um componente (ou sistema) já testada;
- **delta**: modificação feita em um componente (ou sistema) e que ainda não foi testada;
- **configuração delta ("delta build")**: configuração executável do sistema contendo deltas e linhas básicas;
- **casos de teste de regressão**: são os casos testes aplicados à linha base com veredicto = "passou";
- **falha de regressão**: caracterizada quando dois componentes estáveis, um sendo da Linha Básica e outro da Configuração Delta, funcionam adequadamente quando testados sozinhos, mas falham quando usados juntos [BIN99].

4.2 Situações em que os testes de regressão devem ser aplicados

Os testes de regressão devem ser aplicados nas seguintes situações [MAR01]:

- aplicações críticas que devem ser retestadas freqüentemente;
- produto de software que é alterado constantemente durante o desenvolvimento (por exemplo, processo incremental);
- averiguar se componentes reutilizáveis são adequados para o novo sistema;
- durante os testes de integração;
- durante os testes, após as correções;
- na fase de manutenção (corretiva, adaptativa, evolutiva ou preventiva);
- na identificação de diferenças no comportamento do sistema quando há mudanças de plataforma.

Para aplicações orientadas a objeto, os testes de regressão deveriam ser aplicados nas seguintes situações [MAR01]:

- criação de subclasse;

- alteração de superclasse;
- alteração de classe servidora;
- reutilização de uma classe em um novo contexto;
- manutenção dos casos de testes.

4.3 Manutenção dos casos de testes

Depois de várias liberações de versões de *software*, um conjunto de casos de testes pode se tornar muito grande. Sendo assim, alguns casos de teste podem se tornar obsoletos e devem ser removidos. Muitos testadores são relutantes em descartar casos de testes com medo de estarem jogando fora casos que poderiam revelar alguma falha. Contudo, os seguintes tipos de casos de teste podem ser (ou devem ser) descartados [BIN99]:

- **casos de testes "quebrados"**: casos de testes para um componente ou interface que foram removidos ou alterados; logo, os resultados esperados não serão iguais aos resultados obtidos, daí a necessidade de descartar esses casos de testes;
- **casos de testes obsoletos**: os casos de testes não se aplicam mais devido a mudanças nos requisitos (regra de negócio);
- **casos de testes incontroláveis**: os casos de testes são sensíveis a dados de entrada e estados que não estão sob controle. Esses tipos de testes devem ser relegados ao status de "*teste fumaça*", no qual o teste "passa" se a entrada não provoca uma exceção ou término anormal do *software* sob teste;
- **casos de testes redundantes**: dois casos de testes são idênticos se suas entradas e saídas para uma interface específica são idênticas. Casos de testes idênticos não são, necessariamente, redundantes.

4.4 Limitações

Dentre as limitações dos testes de regressão temos [HAR94]:

- uma seqüência de regressão não contém testes para as partes novas ou alteradas;

- uma seqüência de testes que pode ser usada como seqüência de regressão deixa de ser útil como seqüência de testes primária;
- uma seqüência de regressão não tem as mesmas metas de cobertura de uma seqüência de testes primária;
- uso de seqüência de testes inadequada como seqüência de regressão não melhora sua qualidade.

4.5 Automação dos testes de regressão

Aplicar testes de regressão manualmente não é fácil, funciona apenas em estágios iniciais. Para a realização de testes manuais, existem apenas duas alternativas, que são, focar no incremento final ou aumentar a equipe de testes para reexecutar todos os testes de regressão. Devido a custos, geralmente a primeira opção é a escolhida [BIN99].

Para que a automação dos testes de regressão seja realizada, alguns requisitos deverão ser satisfeitos, conforme são mostrados a seguir.

4.5.1 Requisitos para automação dos testes de regressão

Para tornar os testes de regressão automatizados, são definidos os requisitos que são desejáveis, como: controle de versão, estrutura modular, comparar resultados do *baseline* e delta, comparadores inteligentes, embaralhamento, testes built-in [BIN99]. Abaixo, eles são descritos em detalhes:

- **controle de versão**: os conjuntos de testes e o *software* em teste devem estar sob um controle de configuração. Esse controle consiste em um conjunto de atividades projetadas para controlar modificações, identificando os produtos de trabalho que podem ser modificados, estabelecendo relações entre eles, definindo mecanismos para administrar as diferentes versões desses produtos de trabalho, controlando as modificações impostas, fazendo auditoria, e preparando relatórios sobre modificações efetuadas [PRE02];
- **estrutura modular**: os conjuntos de testes *baseline* crescem com incrementos sucessivos e liberações de versões. Organizar os casos de testes em pequenos módulos que correspondem aos casos de uso,

componentes ou outra unidade de trabalho irá facilitar a manutenção. Em outras palavras, um conjunto de testes monolíticos pode ser difícil de usar e vulnerável a erros;

- **comparar resultados do *baseline* e *delta***: os resultados esperados para cada caso de teste *baseline* devem ser salvos numa forma que possam ser comparados com os resultados obtidos para cada caso de teste de regressão. Avaliações automáticas de "passou" ou "não passou" não podem ser obtidas sem comparar os resultados *delta* e *baseline*. O uso de resultados esperados gerados para um caso de teste *baseline* ou resultados obtidos produzidos para esses testes dependem da aplicação e do projeto de testes;
- **comparadores inteligentes**: a habilidade de ignorar alguns campos de saída enquanto compara resultados esperados com resultados obtidos pode ser útil. Por exemplo, suponhamos que o software sob teste tenha como saída um "time stamp" que necessariamente mude a cada execução. Comparar "par de igualdades" iria resultar em "não passou", mesmo que todos os outros campos de saída fossem os mesmos. Essa situação pode ser evitada se a comparação puder ignorar esse campo e garantir apenas que a data seja válida;
- **embaralhamento**: alguns erros são sensíveis à seqüência. Se os testes *baseline* que independem de seqüência são sempre reexecutados na mesma ordem, será perdida a oportunidade de revelar erros de seqüência. A probabilidade de revelar esses tipos de erros aumenta se os testes são "embaralhados" na sua ordem de execução;
- ***testes built-in***: assertivas podem checar se ambos, cliente e servidor obedecem ao uso do contrato.

4.5.2 Ferramentas para automação dos testes de regressão

Essas ferramentas geralmente geram roteiros de testes baseando-se na gravação das atividades dos usuários. Uma vez gravadas as execuções das aplicações, elas podem ser repetidas quando necessário. Exemplos de ferramentas desse tipo podem ser consultados em [HUN01] [DIA01].

4.6 Ambiente de execução dos testes de regressão

Idealmente, os testes de regressão deveriam ser executados num ambiente idêntico ao que foi utilizado nos testes da versão *baseline*. Na prática, isso nem sempre é possível, devido aos seguintes fatores:

- uma diferença ocorre no ambiente de teste – por exemplo, no hardware, no sistema operacional, interface gráfica, SGBD, compilador, linkeditor, utilitários, ferramentas de teste, e assim por diante;
- conteúdo diferente em uma base de dados usada pelo software sob teste – por exemplo, arquivos, bancos de dados, e assim por diante. Restaurar o mesmo estado anterior pode ser difícil ou impossível quando testadores devem usar sistemas existentes que são mantidos em outras partes, ou que são parte de uma aplicação em funcionamento;
- o software sob teste usa objetos não-determinísticos - por exemplo, geradores de coleção que não garantem que os membros da coleção são sempre retornados na mesma ordem;
- o software sob teste usa geração pseudo-randômica – por exemplo, o relógio do sistema é usado como início do processo (semente);
- o software sob teste é sensível a diferenças de tempo na carga ou proporção de chegada de alguns dados de entrada determinados por um agente externo não controlado pelo sistema;
- o software sob teste aceita transações que requerem serialização para resultados consistentes, mas serialização não é suportada, não habilitada, ou é provido por uma parte de desenvolvimento que está atrasada;
- o software sob teste usa código sensível à hora e à data;
- o software sob teste usa origens de entrada não controláveis – por exemplo, sinais de processos físicos, linhas de comunicação, tráfego atual de e-commerce, etc.
- obter uma configuração de testes pode ser muito difícil em um ambiente multiplataforma e complexo. Em sistemas distribuídos, muitos desses fatores, na maioria das vezes, não estão sob o controle do testador. Testadores podem tentar identificar fatores não-controláveis e casos de testes

influenciados por eles. Esses testes podem ser removidos do conjunto de casos de testes de regressão ou relegados ao status de “teste fumaça”. Ou seja, desde que não haja interrupções de processamento, o teste é considerado como “passou”. Isso, porque a produção de um resultado diferente não necessariamente revelará uma falha de regressão.

4.7 Técnicas seletivas de testes de regressão

Técnicas de reteste seletivo reduzem o custo dos testes de regressão utilizando os testes já existentes, e identificando partes do programa modificado que devem ser testadas. Com isso, apenas os casos de testes que exercitam essas partes deverão ser reexecutados. Isso reduz o conjunto de testes a ser aplicado durante os testes de regressão [HAR97].

O que as técnicas de reteste seletivo fazem é dividir o conjunto original de testes em dois subconjuntos, um que será utilizado durante os testes de regressão e outro que não será. Para reduzir o tempo envolvido na execução dos testes, é importante selecionar apenas aqueles testes que produzirão saídas diferentes em relação às saídas originais produzidas pelo programa original, quando esses forem utilizados na versão modificada do programa [HAR94]. Leung and White, porém, mostram que utilizar uma técnica de reteste seletivo só vale a pena, se o custo de implementar tal técnica for menor do que o custo para executar aqueles testes excluídos pela técnica [HAR97].

4.8 Critérios de classificação das técnicas

Binder em [BIN99] ao analisar a forma como as técnicas seletivas de testes de regressão reduzem o conjunto de testes a ser reaplicado, classifica-as em seguras e inseguras. São consideradas seguras apenas as técnicas que, ao reduzirem o conjunto de testes a ser reaplicado, levam em consideração a análise de dependência de código. As técnicas que utilizam outras formas de redução do conjunto de testes são consideradas inseguras.

Além disso, as técnicas de regressão podem ser avaliadas segundo alguns

critérios, que são, inclusão, precisão, eficiência, generalidade e suporte à cobertura. Estes critérios são discutidos nessa seção [HAR94] [MAR01].

Para demonstrar como esses critérios funcionam, vamos continuar assumindo que *P* é um programa, *P'* é a versão modificada de *P*, *T* é um conjunto de testes aplicados a *P*, *S* é uma técnica de reteste seletivo, e *T'* é um conjunto de testes selecionados por *S* dados *P*, *P'* e *T*.

4.8.1.1 Inclusão

Mede o quanto a técnica seleciona testes que irão fazer com que o programa modificado produza diferentes saídas em relação ao programa original.

Suponha que *T* contém n testes *reveladores de modificações*, e *S* seleciona *m* desses testes. A *inclusão* de *S* relativa a *P*, *P'* e *T*, é o percentual calculado pela expressão ((*m*/*n*) *100).

Por exemplo, se *T* contém 50 testes, e 8 desses testes são *reveladores de modificações*, e *S* seleciona apenas 2 desses 8 testes, então *S* é 25% *inclusiva* em relação a *P*, *P'* e *T*.

Quando uma técnica seletiva *S* sempre seleciona todos os testes reveladores de modificações, dizemos que *S* é uma *Técnica Segura* [HAR94] [HAR96].

Definição: Se para todo *P*, *P'* e *T*, *S* é 100% *inclusiva* em relação a *P*, *P'* e *T*, então *S* é uma *Técnica Segura* [HAR94] [HAR96].

Nesse ponto, percebe-se que existem duas definições de “técnicas seguras”, a primeira quanto à forma de redução do conjunto de testes a ser reaplicado durante os testes de regressão; e a outra, quanto ao critério de inclusão dos testes reveladores de modificação no subconjunto de testes a ser reaplicado. Portanto, uma técnica pode ser avaliada como segura porque leva em consideração a análise de dependência no momento da redução do conjunto de testes, e também, porque consegue reunir nesse subconjunto todos os casos de testes reveladores de modificação.

4.8.1.2 Precisão

Mede a habilidade da técnica em não selecionar os testes que não são

reveladores de modificações. A *precisão* sempre é em relação a um programa, sua versão modifica e um conjunto de testes.

Suponha que *T* contém *n* testes *não reveladores de modificações*, e *S* seleciona *m* desses testes. A *precisão* de *S* relativa a *P*, *P'* e *T*, é o percentual calculado pela expressão ((m/n) * 100).

Por exemplo, se *T* contém 50 testes, 44 desses testes são *não reveladores de modificações* com respeito a *P'*, e *S* não seleciona 33 desses 44 testes, então *S* é 75% *precisa* em relação a *P*, *P'* e *T*.

4.8.1.3 Eficiência

Mede o custo computacional da técnica. Esse critério se preocupa basicamente em saber:

- **espaço**: quanto de informação o algoritmo necessita?
- **tempo**: qual a complexidade do algoritmo?

4.8.1.4 Generalidade

Preocupa-se em medir o quanto a técnica é genérica. Basicamente a técnica se preocupa em responder às seguintes questões:

- A técnica permite tratar qualquer tipo de modificação?
- A técnica pode tratar diferentes linguagens e tipos de programas?
- A técnica funciona tanto para testes de unidade quanto para testes de integração?

4.8.1.5 Suporte à cobertura

Preocupa-se com a cobertura da técnica em relação a algum critério.

- Os critérios de cobertura usados para gerar T continuam sendo satisfeitos?
- A técnica permite que seja obtido T' que cubra os acréscimos ou as modificações?

4.9 Técnicas de teste de regressão

Quando falamos em aplicar testes de regressão, as seguintes técnicas podem ser selecionadas: “Retestar tudo”, “Retestar casos de uso de maior risco”, “Retestar por perfil”, “Retestar código modificado” e “Retestar dentro do *firewall”* [BIN99]. A seguir é dada uma visão geral das técnicas. Para obter maiores detalhes sobre elas, o trabalho de Binder poderá ser consultado.

4.9.1 Retestar tudo

Consiste na reexecução de todos os testes *baseline*. Portanto, trata-se de um processo caro, principalmente em função do tempo necessário para a aplicação deste conjunto de casos de teste.

Essa técnica pode ser avaliada, segundo os critérios de inclusão, precisão, eficiência e generalidade, da seguinte forma:

- quanto ao critério de inclusão*,* a técnica propicia a inclusão de todos os casos de testes reveladores de modificações, portanto a mesma é avaliada como *segura*;
- é a técnica menos precisa, pois seleciona todos os casos de testes que deveriam ser omitidos durante os testes de regressão;
- quanto à eficiência, possui o menor custo de análise e configuração, porém possui o maior custo de execução, por ter que executar todos os casos de testes *baseline*;
- quanto à generalidade, a técnica pode ser empregada a qualquer instante e em qualquer escopo.

4.9.2 Retestar casos de uso de maior risco

Consiste na seleção dos testes *baseline* para reexecutar baseando-se em heurísticas de risco. Dentre as funcionalidades (casos de uso) existentes, devem ser selecionadas para o reteste, aquelas que são consideradas mais críticas.

A técnica pode ser avaliada, segundo os critérios de inclusão, precisão, eficiência

e generalidade, da seguinte forma:

- quanto ao critério de inclusão*,* a técnica é avaliada como insegura, pois não leva em consideração a análise de dependência na hora de selecionar os casos de testes;
- sua precisão não é garantida, pois pode selecionar casos de testes que deveriam ser omitidos durante os testes de regressão;
- a eficiência da técnica é difícil de ser medida, a menos que se tenha disponível uma análise de custo e tempo gastos durante os testes do *baseline*;
- quanto à generalidade, a mesma pode ser empregada a qualquer instante e em qualquer escopo;
- é considerada uma técnica insegura, por não levar em consideração a análise de dependência de código na hora de selecionar os casos de testes que serão reexecutados [BIN99].

4.9.3 Retestar por perfil

Consiste na seleção dos testes *baseline* para reexecutar baseando-se na alocação de tempo e por freqüência de uso. Dentre os casos de uso existentes e sua freqüência de utilização, a técnica dá ênfase a aqueles casos cuja freqüência de uso é maior.

A mesma pode ser avaliada, segundo os critérios de inclusão, precisão, eficiência e generalidade, da seguinte forma:

- quanto ao critério de inclusão*,* a técnica é avaliada como insegura, pois não leva em consideração a análise de dependência na hora de selecionar os casos de testes;
- sua precisão não é garantida, pois pode selecionar casos de testes que deveriam ser omitidos durante os testes de regressão;
- a eficiência da técnica é difícil de ser medida, a menos que se tenha disponível uma análise de custo e tempo gastos durante os testes do *baseline*;

- quanto à generalidade, a mesma é mais bem aplicada ao nível de testes de sistemas, quando o escopo é pequeno sua efetividade diminui;
- é considerada uma técnica insegura, por não levar em consideração a análise de dependência de código na hora de selecionar os casos de testes que serão reexecutados [BIN99].

4.9.4 Retestar código modificado

Consiste na seleção dos testes *baseline* que executam os trechos modificados entre as versões *baseline* e *delta*.

Para o uso da técnica, devem ser utilizadas ferramentas para controle de versões além de ferramentas que possam associar trechos de código a casos de testes que os exercitam. Através da comparação entre as versões *baseline* e *delta*, são observadas as modificações entre as mesmas. Para os trechos modificados encontrados, são então selecionados através da associação entre trechos de código e casos de testes, aqueles casos de testes que os exercitam.

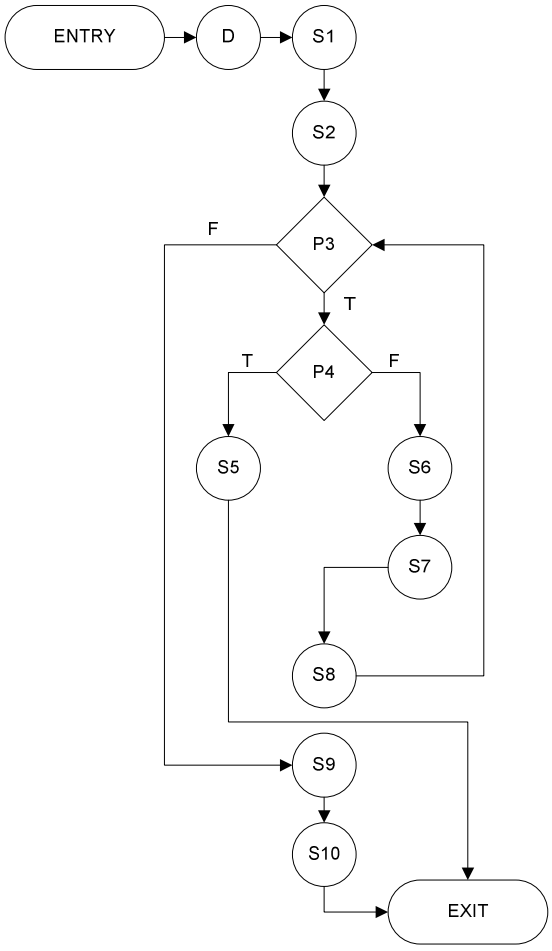
A técnica pode ser avaliada, segundo os critérios de inclusão, precisão, eficiência e generalidade, da seguinte forma:

- quanto ao critério de inclusão*,* a técnica é avaliada como segura, pois leva em consideração a análise de dependência na hora de selecionar os casos de testes [BIN99];
- sua precisão é a maior dentre as técnicas que levam em consideração o código fonte, pois são selecionados todos os casos de testes que exercitam os trechos modificados, sendo excluídos todos aqueles que não os exercitam;
- a técnica não garante nenhuma redução de tempo e custo de execução. Além disso, as ferramentas de análise de cobertura de código, a qual relaciona os casos de testes aos trechos de código, e as de controle de versões devem ser incluídas como custos indiretos;
- quanto à generalidade, a mesma exige que a equipe tenha conhecimento em análise de dependência e ferramentas relacionadas.

Um uso da técnica é descrito por Harrold, através da implementação de um

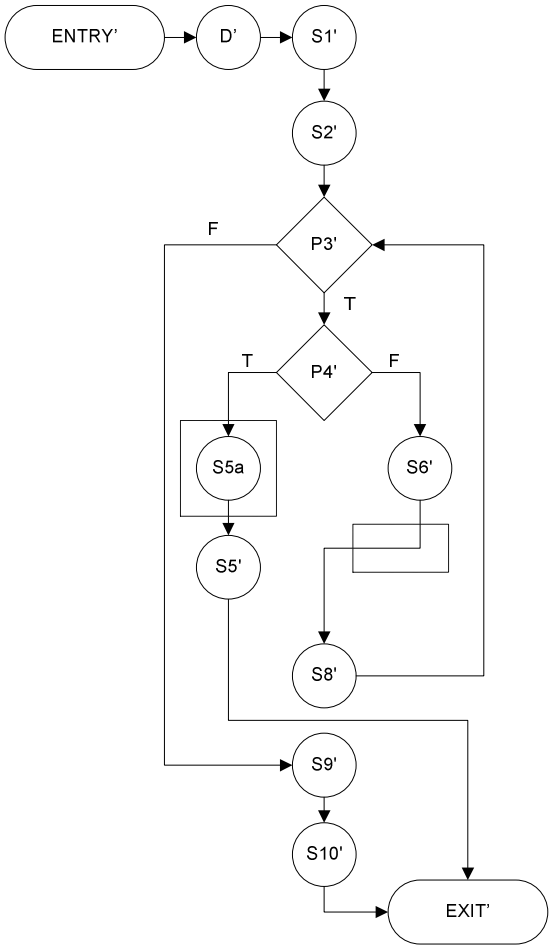
algoritmo seguro de seleção de casos de testes utilizando a técnica de reteste de código modificado. O algoritmo, na sua essência, consiste na execução dos seguintes passos [HAR97]:

- geração de grafos de fluxo de controle para as versões originais e modificadas do software sob teste. As figuras 4.1 e 4.2 exibem um exemplo de grafos de fluxo de controle gerada para as versões original e modificada de um procedimento hipotético chamado AVG;
- comparação dos grafos de fluxo de controle das versões original e modificada dos componentes, visando identificar os trechos modificados entre as duas versões;
- seleção dos casos de testes que exercitam os trechos modificados encontrados. Para isso, podem ser montadas tabelas como a 4.1 e a 4.2, em que a primeira relaciona para cada caso de teste, os arcos de um programa/procedimento exercitados por ele. A segunda tabela relaciona os arcos do programa/procedimento aos casos de testes existentes. Nessa segunda tabela, para cada arco é montado um “vetor de bits”, onde o primeiro bit corresponde ao caso de teste 1, o segundo bit ao caso de teste 2, e assim por diante. Nesse vetor de bits, são marcos com o valor “1”, aqueles casos de testes que exercitam o respectivo arco, e com “0”, aqueles que não o exercitam. Ao final da execução destes passos para cada componente modificado, tem-se em mãos, o conjunto de casos de testes a ser utilizado durante os testes de regressão.




```
                    Procedure avg
S1.     count = 0
S2.     fread(fileptr,n)
P3.     While (not EOF) do
P4.       if (n>0)
S5.             return (error)
          else
S6.         numarray[count] = n
S7.         count++
          endif
S8.       fread(fileptr,n)
        endwhile
S9.     avg = calcavg(numarray, count)
S10.    return avg
```

Figura 4.1 Procedimento AVG e seu grafo de fluxo de controle [HAR97]




```
                    Procedure avg
S1'.      count = 0
S2'.      fread(fileptr,n)
P3'.      While (not EOF) do
P4'.        if (n>0)
S5a.                print ('bad input')
S5'.                return(error)
                  else
S6'.               numarray[count] = n

                  endif
S8'.        fread(fileptr,n)
          endwhile
S9'.      avg = calcavg(numarray, count)
S10'.     return avg
```

Figura 4.2 Procedimento exemplo AVG com modificações [HAR97]

Tabela 4.1 Histórico de testes para o procedimento AVG [HAR97]

Caso Teste	**Entrada**	**Saída**	**Histórico Execução**
T1	Arquivo vazio	0	(entry, D), (D, S1), (S1, S2), (S2, P3), (P3, S9), (S9, S10), (S10, exit)
T2	-1	Erro	(entry, D), (D, S1), (S1, S2), (S2, P3), (P3, P4), (P4, S5), (S5, exit)
T3	1 2 3	2	(entry, D), (D, S1), (S1, S2), (S2, P3), (P3, P4), (P4, S6), (S6, S7), (S7, S8), (S8, P3), (P3, S9), (S9, S10), (S10, exit)

Tabela 4.2 Relação entre arcos e casos de testes que os exercitam [HAR97]

Arco	**Casos de Testes (T1 a T3) "TestsOnEdge"**
(entry, D)	111
(D, S1)	111
(S1, S2)	111
(S2, P3)	111
(P3, P4)	011
(P3, S9)	101
(P4, S5)	010
(P4, S6)	001
(S5, exit)	010
(S6, S7)	001
(S7, S8)	001
(S8, P3)	001
(S9, S10)	101
(S10, exit)	101

4.9.5 Retestar dentro do *firewall*

Consiste na seleção dos testes *baseline* para reexecutar através da análise de dependências de código.

A técnica é descrita por Binder para ser utilizada durante os testes de aplicações orientadas a objetos [BIN99].

Firewall é um conjunto de componentes cujos casos de testes deverão ser selecionados durante os testes de regressão. Dado um conjunto de objetos que foram modificados, o *firewall* compreende o conjunto de módulos que deverão ser retestados. O *Firewall* é uma fronteira imaginável que delimita a parte do software que pode vir a falhar devido às modificações efetuadas. Ele é identificado pela análise das mudanças em cada componente do software sob teste e suas dependências com outros componentes. A tabela 4.3 exibe as regras de seleção dos testes. Cada par de componentes, A e B, é analisado. A alteração ocorrida ou é uma alteração de contrato ou de implementação.

Uma alteração de contrato altera a interface externa do componente, e consiste na adição ou remoção dos atributos que são visíveis externamente, como por exemplo, métodos públicos. Já as alterações de implementação são aquelas que são visíveis apenas ao próprio componente, não alterando a forma como os outros componentes o "vêem".

4.9.5.1 Procedimento

Identificar, desenvolver e executar o conjunto de testes reduzido. Para isso, os seguintes passos deverão ser seguidos:

- criar um matriz de dependência entre as classes do software sob teste. Através dessa matriz, são expressos os relacionamentos de: "A é cliente de B", "B usa A", "B é uma subclasse de A", "B é servidor de A", e assim por diante. Um exemplo dessa matriz é mostrado na tabela 4.4.

  aplicar as regras de decisão da tabela 4.3 para cada célula. Por exemplo, suponha que apenas a implementação da classe "Dinheiro" foi alterada. Quais casos de teste devem ser reexecutados? Pela tabela, vê-se que "Conta" e "Transação" dependem de "Dinheiro". Aplicando a tabela de decisão temos então:

  - nenhuma modificação para "Conta", mudança de implementação para "Dinheiro": Nível 3 de reuso para os casos de testes de "Conta". Nível 2 para os casos de testes de "Dinheiro", e todos os testes de integração aplicáveis a "Conta" e "Dinheiro".
  - nenhuma mudança para "Transacao", mudança de implementação para "Dinheiro": Nível 3 de reuso para os casos de testes de "Transação". Nível 2 de reuso para os casos de testes de "Dinheiro", e todos os testes de integração aplicáveis a "Conta" e "Dinheiro".
- criar uma lista de todos os casos de testes para cada classe do software sob

Tabela 4.3 Tabela de decisão para seleção dos testes dentro do *firewall* [BIN99]

Modificações		**Dependência B para A**	**Requisitos de Reteste**						
A	**B**		**A**	**A Sub**	**A Super**	**B**	**B Sub**	**B Super**	**AB**
Implementação	Nenhuma	B usa A	2			3	3	4	2
		B subClasse de A	2		4	3	3		4
		B sobrepõe A	2	2					2
		B é servidor de A	2						
Contrato	Nenhuma	B usa A	1			2	3	4	1
		B subClasse de A	1	2	4				1
		B sobrepõe A	1	1					1
		B é servidor de A	0						0
Nenhuma	Implementação	B usa A	3	3		2			2
		B subClasse de A	3	3		2		4	3
		B sobrepõe A	2	3		2			3
		B é servidor de A	3			2			2
Nenhuma	Contrato	B usa A	3	3		1			1
		B subClasse de A	3	3		1			?
		B sobrepõe A	1	1		1			1
		B é servidor de A	2			0			0

Legenda:

- **Nível 0**: Nenhum caso de teste pode ser reaproveitado.
- **Nível 1**: os testes para as seqüências de execuções que deixam o sistema no estado desejado podem ser reaplicados sem modificações. Os resultados esperados deverão ser refeitos. A seqüência de execução dos testes nesse conjunto pode ser refeita.
- **Nível 2**: os testes para as seqüências de execuções que deixam o sistema no estado desejado podem ser reaplicados sem modificações. A seqüência de execução dos testes nesse conjunto pode ser refeita.
- **Nível 3**: Os testes podem ser reexecutados sem modificações.
- **Nível 4**: Não há necessidade de reteste.

teste. Transcrever o nível de teste encontrado no passo anterior para cada caso de teste das respectivas classes. A tabela 4.5 exibe um exemplo desse passo sendo aplicado ao teste de integração entre as classes “Conta X Dinheiro” e “Transacao X Dinheiro”.

Tabela 4.4 Matriz de dependência para análise do *firewall* [BIN99]

	ServicoFinanceiro	Conta	Transacao	NumroConta	Dinheiro	Tarifa	Lista
ServicoFinanceiro		C	C				
Conta				C	C		
Transacao					C	C	
NumeroConta							
Dinheiro							
Tarifa							G
Lista							

Legenda:

- **C** = cliente; B é cliente de A;
- **A** = associação; B tem um ponteiro para A;
- **U** = usa; B usa A em um argumento;
- **G** = genérico; B é um genérico e usa A como um tipo de parâmetro;
- **I** = herança; B é uma subclasse de A;
- **P** = Polimorfismo; B sobrepõe um método de A;
- **O** = objeto; B é servidor de A.

4.9.5.2 Oráculo

Os resultados obtidos durante a execução do conjunto de casos existente (*baseline*), após passarem pelo processo de manutenção dos casos de testes, são o próprio oráculo, ou seja, se o software sob teste conseguir repetir os mesmos resultados obtidos anteriormente, significa que os testes passaram e o software está correto.

Tabela 4.5 Seleção dos casos de testes dentro do *firewall* [BIN99]

Casos de Testes	Casos de testes selecionados pela dependência	
	Conta, Dinheiro	Transacao, Dinheiro
TesteServicoFinanceiro		
TesteConta	3	
TesteTransacao		3
TesteNumeroConta		
TesteDinheiro	2	2
TesteTarfia		
TesteLista		

Legenda:

- **Nível 0**: Nenhum caso de teste pode ser reaproveitado.
- **Nível 1**: os testes para as seqüências de execuções que deixam o sistema no estado desejado podem ser reaplicados sem modificações. Os resultados esperados deverão ser refeitos. A seqüência de execução dos testes nesse conjunto pode ser refeita.
- **Nível 2**: os testes para as seqüências de execuções que deixam o sistema no estado desejado podem ser reaplicados sem modificações. A seqüência de execução dos testes nesse conjunto pode ser refeita.
- **Nível 3**: Os testes podem ser reexecutados sem modificações.
- **Nível 4**: Não há necessidade de reteste.

  teste. Transcrever o nível de teste encontrado no passo anterior para cada caso de teste das respectivas classes. A tabela 4.5 exibe um exemplo desse passo sendo aplicado ao teste de integração entre as classes “Conta X Dinheiro” e “Transacao X Dinheiro”.

4.9.5.3 Pré-requisitos

Para início da aplicação da técnica, os componentes já deverão ter sido passados por testes unitários; o conjunto de testes do *baseline* já deverá estar disponível e o ambiente deverá ter sido restaurado conforme estava durante a execução dos testes *baseline*. Caso essa restauração não seja possível, aqueles testes que dependem do ambiente não poderão ser avaliados com total segurança.

4.9.5.4 Critérios de parada

Para terminar a execução dos testes, após a correção das falhas encontradas, deverão ainda existir apenas aquelas cuja criticalidade é aceitável.

4.9.5.5 Avaliação

Quanto ao critério de inclusão, a técnica é avaliada como segura, pois leva em

consideração a análise de dependência na hora de selecionar os casos de testes [BIN99].

Sua precisão é alta, pois poucos casos de testes que poderiam ser omitidos são selecionados.

A técnica não garante nenhuma redução de tempo e custo de execução. Além disso, as ferramentas de análise de cobertura de código, a qual relaciona os casos de testes aos trechos de código, e as de controle de versões devem ser incluídas como custos indiretos.

Quanto à generalidade, a técnica exige que a equipe tenha conhecimento em análise de dependência e ferramentas relacionadas.

4.10 Critérios de comparação entre as técnicas

Uma visão bem prática de comparação entre as técnicas existentes consiste em compará-las em relação aos seus respectivos custos e riscos. A tabela 4.6 exibe tal comparação, nela são listadas verticalmente as técnicas de regressão discutidas e, na horizontal, as tarefas envolvidas no testes de regressão de software. Para cada técnica então, são avaliados o custo e risco para cada atividade [BIN99].

Pela análise da tabela, podemos concluir, por exemplo, que a técnica “Retestar Tudo” possui o menor risco de omitir algum caso de teste durante os testes de regressão, porém, possui o maior custo de execução. Além disso, a mesma é considerada segura, por selecionar todos os testes *reveladores de modificações* [HAR94].

Também são consideradas seguras, as técnicas “Retestar Dentro do *Firewall*” e “Retestar Código Modificado”, isso se deve ao fato, das mesmas levarem em consideração a análise de dependência de código no momento da redução do conjunto de testes a ser reaplicado [BIN 99].

Uma outra vantagem que percebemos nas técnicas “Retestar Dentro do *Firewall*” e “Retestar Código Modificado” em função das mesmas serem “seguras”, é o fato das mesmas possuírem uma baixa probabilidade de omitirem casos de testes reveladores

de modificações.

Tabela 4.6 Custo e risco para as técnicas de regressão [BIN99]

	Custo e Risco das Técnicas de Seleção dos Testes de Regressão				
Atividade do Teste de Regressão	**Retestar Tudo**	**Retestar casos de uso de maior risco**	**Retestar por Perfil**	**Retestar Dentro Do *Firewall***	**Retestar Código Modificado**
Remover casos de testes quebrados	~~C~~	~~C~~	~~C~~	~~C~~	~~C~~
Analisar dependências				~~C~~-$	~~C~~-$
Remover casos de testes obsoletos				~~C~~ -$$	~~C~~-$$
Remover casos de testes redundantes				~~C~~-$$	~~C~~-$$
Configurar o ambiente de execução de execução dos testes	~~CC~~	~~C~~	~~C~~	~~C~~	~~C~~
Custo para projetar os casos de testes	~~C~~	~~CC~~	~~CC~~	$$$	$$
Custo para executar os casos de testes	Mais alto	Orçamento limitado	Orçamento limitado	Tamanho do *Firewall*	Tamanho do Delta
Avaliar testes que não passaram	~~CC~~	~~C~~	~~C~~	~~C~~	~~C~~
Risco de omissão	Mais baixo	Moderado	Moderado	Baixo	Baixo
“Segura”?	Sim	Não	Não	Sim	Sim

Legenda:

- ~~C~~ = menor custo;
- ~~cc~~ = custo baixo;
- $ = custo médio;
- $$ = custo alto.
- $$$ = custo mais alto.

4.11 Considerações finais

Através da análise das técnicas de testes de regressão hoje existentes e citadas

nesse capítulo, percebeu-se que dentre elas, apenas três são consideradas seguras, que são “Retestar Tudo”, “Retestar Código Modificado” e “Retestar Dentro do *Firewall*”.

Vimos que a técnica “Retestar Tudo” é a menos precisa, ou seja, seleciona todos os casos de testes que poderiam ser omitidos. Portanto, exige um maior custo de execução em relação às demais, tornando a sua execução pouco viável quando aplicado a sistemas grandes.

A técnica “Retestar Código Modificado” exige o uso de ferramentas que consigam fazer a associação entre trechos de código e casos de testes, e também ferramentas de controle de versões para fazer a análise de quais modificações foram realizadas.

Foram candidatas então a servirem como suporte à Estratégia Proposta nesse trabalho, as técnicas “Retestar Código Modificado” e “Retestar Dentro do *Firewall”.* Quanto à técnica “Retestar Dentro do *Firewall*”, percebeu-se os seguintes fatores favoráveis ao uso da mesma:

- é segura, pois leva em consideração a análise de dependência entre componentes [BIN99];
- dispensa o uso de um banco de dados que armazena a associação entre casos de testes e trechos de códigos;
- dispensa a análise do código fonte das versões delta e *baseline* dos componentes que sofreram modificações, visando construir seus respectivos grafos de fluxo de controle. Esse fator para aplicações Web é fundamental, pois em virtude da grande heterogeneidade de tecnologias envolvidas na sua implementação, discutidos anteriormente no capítulo 3 “Revisão de Testes de Software”, “Seção 3.7”; realizar essa análise pode ser uma tarefa complicada.

Pelo exposto, e associado ao pouco tempo disponível para se testar aplicações Web, em virtude da sua constante evolução concluiu-se nesse trabalho, que dentre as técnicas estudadas, a técnica “Reteste Dentro do *Firewall*” é a mais viável de ser implantada e, portanto, será a empregada na formulação da estratégia proposta, a qual é detalhada no capítulo seguinte [HUN01].

Capítulo 5 - ESTRATÉGIA PROPOSTA

Para a seleção dos casos de testes a serem aplicados durante os testes de regressão é proposta uma estratégia, que é descrita em detalhes neste capítulo. Serão descritos seus objetivos, particularidades, procedimentos de execução, pré-requisitos, critérios de finalização, possibilidade de automação, bem como uma avaliação da mesma em relação aos critérios de inclusão, precisão, eficiência e generalidade.

5.1 Objetivos da estratégia

O objetivo principal da estratégia consiste em dada uma modificação no software identificar quais casos de testes deverão fazer parte dos testes de regressão. Para isso, sua formulação levou em consideração os seguintes aspectos:

- a estratégia deve ser seletiva, ou seja, durante os testes de regressão não serão reutilizados todos os casos de testes existentes, e o custo-benefício para seleção do conjunto de testes a reutilizar deve ser favorável. Nesse caso, é considerada favorável, uma estratégia de seleção cujo custo para selecionar os casos de testes seja menor do que o de executar os casos de testes excluídos pela mesma [HAR97];
- a estratégia deve ser "segura", ou seja, será baseada na dependência de código dos componentes modificados em relação aos outros componentes que compõem o software sob teste [BIN99]. Com isso, consegue-se selecionar um percentual significativo dos casos de "testes reveladores de modificações", ou seja, aqueles capazes de fazer com que o programa modificado produza diferentes resultados em relação ao original [HAR94].

5.2 Suporte à estratégia

Dentre as estratégias seguras estudadas, "Reteste de Código Modificado" e "Reteste Dentro do *Firewall*" [BIN99], será escolhida a segunda para dar suporte à

estratégia proposta. Tal escolha foi realizada pela facilidade de implementação da estratégia, e pelo seu custo de implantação. Pela comparação das duas estratégias, a segunda apresentou algumas vantagens, as quais são mostradas na tabela 5.1. Tais vantagens implicam diretamente na redução do trabalho e custo envolvidos, o que torna a estratégia "Reteste Dentro do *Firewall*" mais barata e simples.

Tabela 5.1 Vantagens da técnica de reteste dentro do *firewall* em relação à de reteste do código modificado

Vantagens
Não há necessidade de se manter um histórico de execução para cada caso de teste. O histórico de execução consiste nas instruções do programa que são exercitadas para cada caso de teste
Não há necessidade de se comparar os códigos fontes dos programas modificado e original, visando encontrar as modificações realizadas, e a partir daí encontrar os casos de testes que exercitam essas modificações.

5.3 Particularidades da estratégia

A estratégia procura auxiliar não somente os testes de regressão, mas também a manutenção de sistemas. Para isso, ela propõe algumas rastreabilidades (associações), que permitem "caminhar" entre os diferentes níveis de abstração do sistema, como por exemplo, a partir dos casos de uso, chegar aos componentes que os implementam. O que torna mais fácil o entendimento do sistema. Além disso, o conceito de componentes é amplamente utilizado na estratégia, o que não acontece na estratégia de original de "Reteste Dentro do *Firewall*" [BIN99] descrita na "seção 4.9.5" do capítulo "4 Testes de Regressão", no qual são tratadas classes ao invés de componentes. Para a estratégia, componentes são mapeados em páginas Web e demais programas associados a elas como, por exemplo, programas COBOL que ficam no computador central (mainframe), e são acessados via páginas Web.

5.3.1 Web x sistemas orientados a objetos

A técnica original de "Reteste dentro do *Firewall*" descrita em [BIN99] é empregada a sistemas orientados a objetos e, portanto, a análise de dependências entre os componentes leva em consideração relacionamentos de herança, associação

ou agregação.

Como a estratégia proposta não é empregada a sistemas orientados a objetos, e sim a aplicações Web em geral, as quais nem sempre são orientadas a objetos, os relacionamentos de herança e agregação não serão tratados, e sim, apenas as "chamadas" convencionais entre componentes, como "*call programa"* e *links* entre páginas Web.

Portanto, entre os componentes que compõem as aplicações Web serão analisadas apenas as dependências de "dado um componente, quais outros o usam" e, "dado um componente, quais outros ele usa". Ou seja, serão analisadas as dependências "B é servidor de A" e "B é cliente de A", respectivamente, sendo excluídas as dependências "B é uma subclasse de A" e "B sobrepõe A", discutidas na "seção 4.9.5.1" do capítulo "4 Testes de Regressão".

5.3.2 Rastreabilidade entre casos de uso e componentes

Um dos grandes problemas da manutenção de sistemas é a falta de documentação [PIG97] e, associado a isso, a dificuldade em responder à seguinte pergunta: "Dada uma funcionalidade, quais os componentes que a implementam?" Um mecanismo que auxilie na obtenção dessa resposta irá, dado uma solicitação de manutenção em uma funcionalidade, agilizar a localização dos componentes a serem modificados e, também, facilitar a análise de impacto das manutenções.

Esse problema de localizar no sistema o código fonte responsável por implementar determinada funcionalidade é tratado por Bojic e Delasevic [BOJ00] em seu método proposto de recuperação de arquiteturas de sistemas legados orientados a objetos. Nesse método é feita a associação entre casos de uso e funcionalidades do sistema, o que facilita as manutenções futuras e ajuda a melhorar o conhecimento do sistema.

Na estratégia proposta essa associação entre casos de uso e componentes que os implementam também é empregada.

Caso de Uso: um caso de uso é uma seqüência de ações executadas por um sistema, cujos resultados são visíveis a um determinado ator. Ator é alguém ou algo

que está fora do sistema, mas que interage com ele [KRU00].

A associação entre atores, casos de uso e componentes é descrita em [BIN99]. A figura 5.1 exibe essas associações.

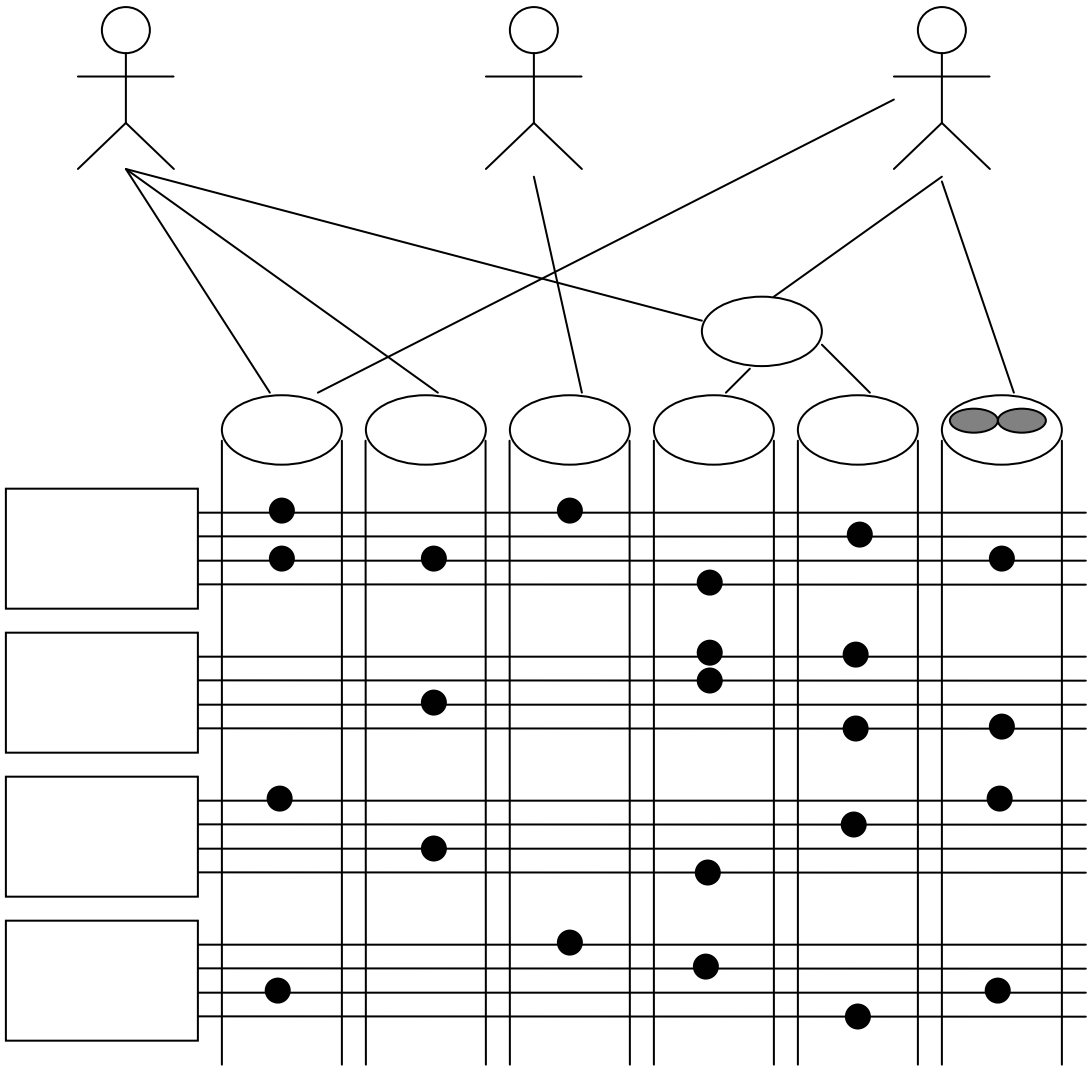


Figura 5.1 Associação entre componentes, atores, casos de uso e componentes [BIN99]

Pela análise da figura, vê-se que um componente (caixas à esquerda na figura) dá suporte a vários casos de uso (ovais no topo da figura), e que um caso de uso é implementado por um ou mais componentes [BIN99].

5.3.3 Rastreabilidade entre casos de uso e casos de testes

Como o número de componentes em um sistema Web tende a ser muito grande, podendo uma única página Web ser formada por vários componentes, entre *frames*, *forms* e *arquivos include*[1], o trabalho de manter a associação entre casos de testes e componentes pode ser muito complicado e trabalhoso. Além disso, um único componente pode ter vários casos de testes e novos casos de testes podem ser criados a cada modificação realizada.

Associado a isso e ao pouco tempo disponível para se testar aplicações Web, devido ao grande número de solicitações de modificações que as mesmas sofrem constantemente, os testes de unidades são praticamente impossíveis de serem aplicados na prática. Como um bom nível de qualidade deve ser atendido para essas aplicações, a estratégia propõe então, que os testes sejam aplicados para casos de uso, o que não impede que os componentes sejam testados, pois são eles os responsáveis pelas implementações dos casos de uso. Um caso de uso é, na maioria das vezes, implementado por diversos componentes, especialmente no modelo de projeto de "n-camadas", que pode ser representado pela separação das camadas de apresentação, regras de negócio e o acesso ao banco de dados de dados, conforme exemplo ilustrado na figura 5.2 [JAC03]. O teste de um caso de uso implicará em testar todos esses componentes, economizando-se tempo e recursos e favorecendo a sua real execução.

5.4 Procedimentos para execução da estratégia

Para conseguirmos selecionar os casos de testes que farão parte dos testes de regressão utilizando a estratégia proposta, três fases deverão ser executadas, que são "Preparação da infra-estrutura", "Obtenção do *firewall*" e "Refinamento dos casos de testes", descritas a seguir:

---

[1] Esses arquivos geralmente possuem rotinas que são reutilizadas, como por exemplo, rotinas de validação de CPF, datas, CNPJ, etc.

5.4.1 Preparação da Infra-Estrutura

A primeira fase da estratégia consiste em montar uma base de dados constituída pelos casos de uso que compõem o software sob teste, seus componentes, casos de testes e os relacionamentos entre eles. Para isso, os passos abaixo deverão ser seguidos:

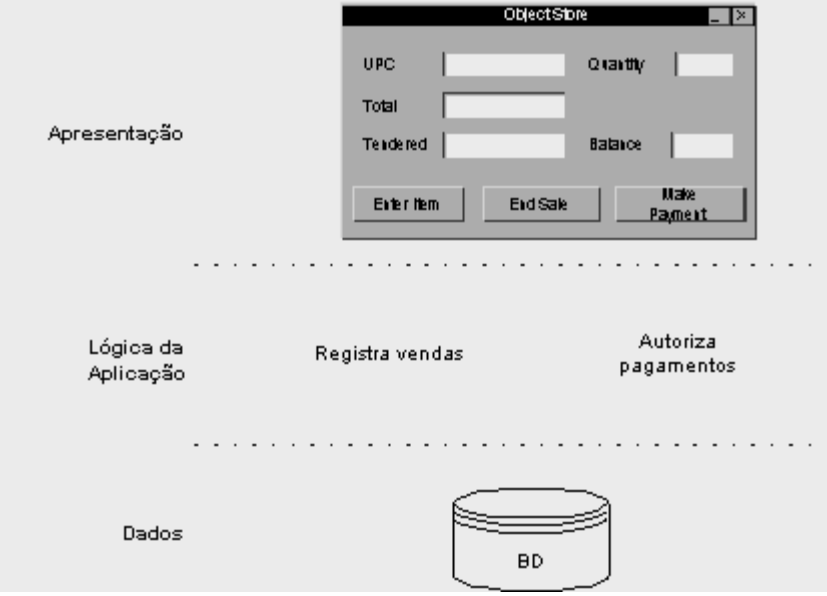


Figura 5.2 Arquitetura de projetos n-camadas

1. relacionar os casos de uso que dizem respeito ao software sob teste. Para isso, pode ser montada uma tabela semelhante à tabela 5.2 mostrada a seguir:

Tabela 5.2 Relação de casos de uso

ID Caso de uso	Descrição do Caso de Uso
UC1	Abrir Conta
UC2	Encerrar Conta
UC3	Efetuar Depósito

2. relacionar os componentes que integram o software sob teste. Para isso, pode ser montada uma tabela semelhante à tabela 5.3.

Tabela 5.3 Relação de componentes

ID Componente	Nome do Componente
C1	EncerrarConta.asp
C2	AbrirConta.asp
C3	ValidarConta.asp
C4	EfetuarDepositoDinheiro.asp
C5	EfetuarDepositoCheque.asp

3. fazer a associação entre os casos de uso e os componentes que integram o software sob teste, obtendo assim uma matriz de dependência entre casos de uso e componentes que os implementam. A tabela 5.4 demonstra essa associação.

Tabela 5.4 Associação entre casos de uso e componentes

Casos de uso	Componentes				
	C1	C2	C3	C4	C5
Abrir Conta		X	X		
Encerrar Conta	X				
Efetuar Depósito			X	X	X

Analisando essa tabela, percebemos que o caso de uso “Abrir Conta” é implementado pelos componentes “C2” (AbrirConta.asp) e “C3” (ValidarConta.asp); o caso de uso “EncerrarConta.asp” pelo componente “C1” (EncerrarConta.asp); e o caso de uso “Efetuar Depósito” pelos componentes “C3” (ValidarConta.asp), “C4” (EfetuarDepositoDinheiro.asp) e “C5” (EfetuarDepositoCheque.asp).

4. relacionar os casos de testes que dizem ao respeito ao software sob teste. Para isso pode ser montada uma tabela semelhante à tabela 5.5.

Tabela 5.5 Relação de casos de testes

ID Caso Teste	Descrição do Caso de Teste
CT1	Testar abrir conta
CT2	Testar encerrar conta
CT3	Testar efetuar depósito em dinheiro
CT4	Testar efetuar depósito em cheque

5. fazer a associação entre os casos de testes e os casos de uso que são exercitados pelos mesmos. Para isso pode ser montada uma tabela semelhante à tabela 5.6 mostrada a seguir.

Tabela 5.6 Associação entre casos de uso e componentes

Casos de uso	Casos de Teste			
	CT1	CT2	CT3	CT4
Abrir Conta	X			
Encerrar Conta		X		
Efetuar Depósito			X	X

Analisando a tabela acima, percebemos que o caso de uso “Abrir Conta” é exercitado pelo caso de teste “CT1” (Testar abrir conta); o caso de uso “Encerrar Conta” pelo caso de teste “CT2” (Testar encerrar conta), e o caso de uso “Efetuar Depósito” pelos casos de testes “CT3” e “CT4” (Testar efetuar depósito em dinheiro e Testar efetuar depósito em cheque, respectivamente).

6. como na técnica original de Binder [BIN99], deve ser montada a matriz de dependência, com a diferença de que para a estratégia proposta, essa dependência deverá ser entre componentes, e não entre classes. Nessa matriz são expressas as dependências “B é servidor de A” e “B usa A”, descritos anteriormente. Para isso, uma tabela como a mostrada abaixo pode ser utilizada. Ela é montada da seguinte forma:
   1. distribua todos os clientes na coluna de “Componentes – Clientes”;
   2. distribua os mesmos componentes agora, como “Componentes – Servidores”;

3. para o preenchimento da Matriz, selecione o primeiro componente “Cliente” e analise se algum outro é servidor para o mesmo. Caso seja, marque um “X” na coluna que faz o cruzamento entre os dois. O mesmo deve ser feito para todos os componentes clientes restantes, tal qual é demonstrado na tabela 5.7 a seguir.

Tabela 5.7 Matriz de dependência entre os componentes

Componentes – Clientes	**Componentes – Servidores**				
	C1	C2	C3	C4	C5
C1					
C2			X		
C3					
C4			X		
C5			X		

Analisando a tabela acima, percebemos que o componente “C3” (ValidarConta.asp) funciona como um servidor para validações de contas e portanto é usado pelos componentes clientes “C2” (AbrirConta.asp), “C4” (EfetuarDepositoDinheiro.asp) e “C5” (EfetuarDepositoCheque.asp).

Com tais informações a infra-estrutura da estratégia proposta está montada e, assim, realizada a primeira fase da mesma. Essa fase da estratégia pode acontecer durante o desenvolvimento do sistema. Caso ela ocorra no desenvolvimento do software sob teste, durante a manutenção do sistema, quando os testes de regressão forem iniciados, essa fase da estratégia poderá ser omitida, podendo ser executada diretamente a segunda fase (apresentada em 5.4.2), que é a obtenção do *firewall*.

5.4.2 Obtenção do *firewall*

O *firewall*, conforme já comentado no capítulo 4, consiste no conjunto de componentes cujos casos de testes deverão fazer parte dos testes de regressão, sendo esse conjunto de componentes identificado através da análise de dependência de código entre os componentes modificados e o restante do sistema [BIN99].

Quando da ocorrência de uma manutenção, para a obtenção do conjunto de testes a ser reutilizado durante os testes de regressão, pela estratégia adotada, os passos a seguir deverão ser executados:

1. obter a informação de quais componentes foram modificados. Isso pode ser feito com a ajuda de um software de controle de configuração, o qual poderá comparar as versões: atual e anterior de cada componente, obtendo assim aqueles que sofreram alguma modificação. Como exemplo, suponhamos que o componente ValidarConta.asp tenha sido modificado. Para listar os componentes modificados, uma tabela pode ser criada tal qual a tabela 5.8 mostrada abaixo.

Tabela 5.8 Relação de componentes modificados durante a manutenção

ID Componente	Nome do Componente
C3	ValidarConta.asp

Como exemplo, para a execução dos passos restantes que compõem a fase "obtenção do *firewall*", será utilizada a tabela acima como base.

2. para cada componente encontrado, através da matriz de dependência entre componentes a qual foi montada na primeira fase da estratégia, selecionar os componentes que são "servidores" e "clientes" do mesmo. No nosso exemplo, para o componente "C3" (ValidarConta.asp), através da análise de dependência, encontramos os componentes "C2", "C4" e "C5" como clientes dele, e como servidor para ele, nenhum componente, conforme se observa na tabela 5.9.

Tabela 5.9 Componentes clientes do componente C3

Componentes – Cliente	**Componentes – Servidor**				
	C1	C2	C3	C4	C5
C1					
C2			X		
C3					
C4			X		
C5			X		

3. para cada componente encontrado no passo anterior através da matriz de dependência entre casos de uso e componentes obtida na fase anterior da estratégia, selecionar os respectivos casos de uso que são implementados pelos componentes encontrados. No exemplo, para o componente “C2” (AbrirConta.asp) é encontrado o caso de uso “Abrir Conta”, e para os componentes “C4” (EfetuarDepositoDinheiro.asp) e “C5” (EfetuarDepositoCheque), o caso de uso “EfetuarDeposito” para ambos. Tal matriz é demonstrada conforme a tabela 5.10 abaixo.

Tabela 5.10 Casos de uso para os componentes dependentes

Casos de uso	**Componentes**				
	C1	C2	C3	C4	C5
Abrir Conta		X	X		
Encerrar Conta	X				
Efetuar Depósito			X	X	X

Deverão ser selecionados também os casos de uso que são implementados pelo componente modificado, no caso “C3”, conforme mostrado na tabela 5.11.

Tabela 5.11 Casos de uso para o componente modificado

Casos de uso	**Componentes**				
	C1	C2	C3	C4	C5
Abrir Conta		X	X		
Encerrar Conta	X				
Efetuar Depósito			X	X	X

Nesse exemplo, o caso de uso “Abrir Conta” já havia sido selecionado pelo componente “C2” (AbrirConta.asp).

4. para cada caso de uso encontrado, selecionar através da matriz de dependência entre casos de uso e casos de testes, obtida na fase anterior da estratégia, os respectivos casos de testes que farão parte do conjunto de casos de testes a ser reutilizado durante os testes de regressão, conforme se demonstra na tabela 5.12.

Tabela 5.12 Casos de testes selecionados para os testes de regressão

Casos de uso	**Casos de Teste**			
	CT1	CT2	CT3	CT4
Abrir Conta	X			
Encerrar Conta		X		
Efetuar Depósito			X	X

Analisando a tabela acima, vemos que o caso de testes “CT1” é selecionado para exercitar o caso de uso “Abrir Conta”, e os casos de testes “CT3” e “CT4” para exercitarem o caso de uso “Efetuar Depósito”.

Ao final desse passo, está concluída a estratégia de seleção dos casos de testes aplicando a estratégia proposta. Portanto, tem-se em mãos o conjunto de casos de testes que compõem o *firewall*, e que, portanto, será reutilizado durante os testes de regressão.

5.4.3 Refinamento dos Casos de Testes

Durante a manutenção alguns componentes sofrem modificações de contrato, ou seja, na sua especificação de serviços fornecidos, ou na sua implementação interna.

Essas modificações implicam na forma como os testes de regressão deverão ser selecionados e aplicados.

Os casos de testes encontrados deverão ser avaliados em relação aos componentes que implementam seus respectivos casos de uso, segundo a tabela de decisão representada pela tabela 5.13, a qual auxilia a execução e atualização dos casos de testes em relação às modificações ocorridas. Essa tabela é um subconjunto da tabela de decisão original da técnica de "Reteste Dentro do *Firewall*" [BIN99]. Para a construção da mesma foram excluídas da tabela original as particularidades de orientação a objetos.

Um exemplo da necessidade de atualização dos casos de testes é o caso de uma página Web ter seus campos editáveis alterados, significando que a mesma teve sua especificação alterada (modificação de contrato). Nessa situação, os casos de testes que exercitam os casos de uso implementados por essa página deverão ser refeitos para comportar tal modificação.

5.5 Oráculo para a estratégia

Os resultados da execução dos testes na versão original após suas devidas atualizações em funções das modificações ocorridas serão utilizados como oráculo [BIN99].

5.6 Pré-requisitos da estratégia

Para ser iniciada a estratégia, a seguinte condições devem ser satisfeitas [BIN99]:

- os testes de unidades já deverão ter sido executados para os componentes da versão modificada do software sob teste;

Tabela 5.13 Tabela de decisão para seleção dos testes dentro do *firewall* para a estratégia

Modificações		**Dependência B para A**	**Requisitos para Reteste**		
A	**B**		**A**	**B**	**AB**
Implementação	Nenhuma	B usa A	2	3	2
		B é servidor de A	2		
Contrato	Nenhuma	B usa A	1	2	1
		B é servidor de A	0		0
Nenhuma	Implementação	B usa A	3	2	2
		B é servidor de A	3	2	2
Nenhuma	Contrato	B usa A	3	1	1
		B é servidor de A	2	0	0

Legenda:

- **Nível 0**: nenhum caso de teste pode ser reaproveitado.
- **Nível 1**: os testes para as seqüências de execuções que deixam o sistema no estado desejado podem ser reaplicados sem modificações. Os resultados esperados deverão ser refeitos. A seqüência de execução dos testes nesse conjunto pode ser refeita.
- **Nível 2**: os testes para as seqüências de execuções que deixam o sistema no estado desejado podem ser reaplicados sem modificações. A seqüência de execução dos testes nesse conjunto pode ser refeita.
- **Nível 3**: os testes podem ser reexecutados sem modificações.
- o conjunto de testes da versão original (antes da manutenção) deverá estar disponível para ser reutilizado;
- a relação dos componentes que sofreram modificação deve estar disponível. Esta relação será utilizada na obtenção do *firewall*.

5.7 Critérios para finalização da execução da estratégia

Os critérios de finalização da estratégia são os seguintes [BIN99]:

- para aqueles componentes que apresentaram falhas, as quais não foram corrigidas, a sua criticalidade não deve impedir o lançamento da nova versão do software sob testes;

- todos os demais testes deverão ter sido executados sem apresentar falhas.

5.8 Automação da estratégia

Pela análise da estratégia, percebeu-se que muitos de seus passos poderiam ser automatizados. No próximo capítulo, então, é apresentada a ferramenta "*FireWeb*", a qual foi desenvolvida para automatizar o uso da estratégia proposta durante os testes de regressão de sistemas Web.

5.9 Considerações Finais

Nesse capítulo, vimos que a estratégia proposta é baseada na técnica de "Retestar Dentro do Firewall" descrita em [BIN99].

Três fases compõem a estratégia proposta, "Preparação da infra-estrutura", "Obtenção do firewall" e "Refinamento dos casos de testes":

- **preparação da infra-estrutura**: tem por objetivo montar as rastreabilidades entre casos de uso, casos de testes e componentes.
- **obtenção do firewall**: tem por objetivo encontrar os componentes os quais deverão ser testados em virtude da modificação ocorrida no software sob testes.
- **refinamento dos casos de testes**: nessa fase, devem ser avaliados os casos de testes já existentes, visando atualizá-los à nova versão do software sob testes.

Foram também apresentadas as particularidades da estratégia, que são, "Web X sistemas orientados a objetos", "rastreabilidade entre casos de uso e componentes" e "rastreabilidade entre casos de uso e casos de testes":

- **Web X sistemas orientados a objetos**: pelo fato de nem sempre as aplicações Web serem orientadas a objeto, as dependências de código entre seus componentes não devem considerar relacionamentos exclusivos de orientação a objetos, como: herança, sobreposição, etc.

- **rastreabilidade entre casos de uso e componentes**: permite identificar quais componentes implementam quais casos de uso.
- **rastreabilidade entre casos de uso e casos de testes:** permite identificar quais casos de testes exercitam quais casos de uso.

Após a obtenção do firewall, o qual é formado pelos componentes que podem sofrer impacto pelas modificações efetuadas no software sob teste, tais rastreabilidades propiciam a obtenção dos casos de testes responsáveis por exercitar tais componentes.

Capítulo 6 - FIREWEB – FIREWALL PARA WEB

Para dar suporte à estratégia proposta no capítulo anterior, e demonstrar a capacidade de automação da mesma, foi construída uma ferramenta implementada em MS Visual Basic 6.0 utilizando banco de dados Access. Nas seções seguintes desse capítulo, serão apresentados seu modelo de dados, principais funcionalidades e limitações dessa primeira versão.

6.1 O modelo de dados da ferramenta

Na figura 6.1 abaixo, é apresentado o modelo de dados responsável por armazenar as informações necessárias à obtenção do firewall.

O mesmo é formado pelas seguintes tabelas:

- modelos Rational Rose (TBModelos);
- casos de uso (TBUseCases);
- casos de testes (TBCasosTestes);
- componentes (TBComponentes);
- associação entre casos de uso e casos de testes TBRastr_UC_CasoTeste);
- associação entre casos de uso e componentes (TBRastr_UC_Componente).

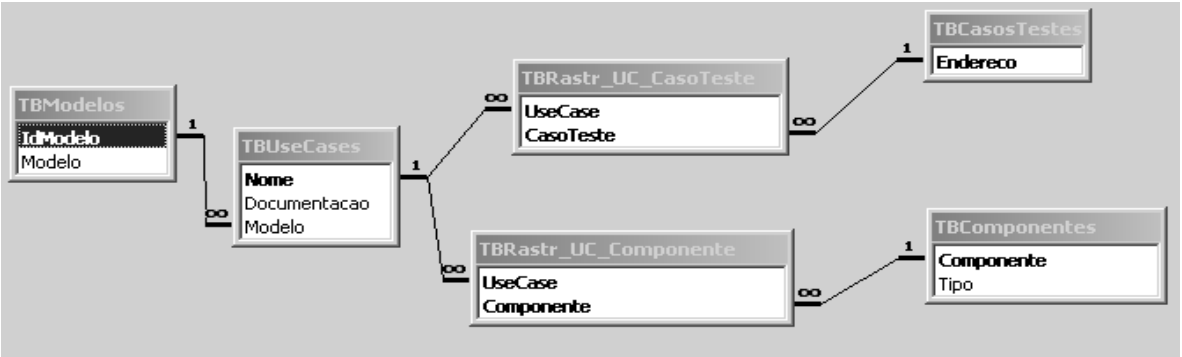


Figura 6.1 Modelo de dados da ferramenta *FireWeb*

O modelo tem como principal objetivo, implementar as rastreabilidades entre:

- casos de uso e casos de testes;
- casos de uso e componentes.

Ambos os relacionamentos são modelados como “NxN” (muitos-para-muitos) [SET89].

6.2 Funcionalidades da ferramenta

A ferramenta é composta por cinco grupos principais de funcionalidades:

- **importação de casos de testes, componentes e casos de uso**. Consiste em armazenar no banco de dados os componentes (apenas o nome), os casos de testes (seus endereços físicos) e os casos de uso (nomes e especificações). Essas importações dão suporte aos passos 1, 2, e 4 da fase “Preparação da Infra-Estrutura” da estratégia proposta, descritos no capítulo anterior, “seção 5.4 – Procedimentos para execução da estratégia”.
- **rastreabilidade**. Utilizado para associar os casos de uso aos componentes, e os casos de uso aos casos de testes. Essas associações dão suporte aos passos 3 e 5 da estratégia proposta, descritos no capítulo anterior, “seção 5.4 – Procedimentos para execução da estratégia”.
- **listas**. Utilizado para consultar o estado atual do banco, ou seja, casos de testes, componentes e casos de uso cadastrados;
- **firewall**. Utilizado para identificar o firewall, o que corresponde à execução da segunda fase da estratégia proposta, “Obtenção do firewall”, descrita na “seção 5.4.2” do capítulo anterior;
- **inicialização de tabelas**. Utilizado para limpar as tabelas do banco, caso seja necessário.

A figura 6.2 exibe o menu principal da ferramenta.

Figura 6.2 Grupos principais de funcionalidades da ferramenta

6.2.1 Importação de casos de testes

A importação de casos de testes consiste em especificar o diretório da máquina onde residem os casos de testes a importar e, por fim, efetuar a importação. São importados para o banco de dados: o caminho completo do local de armazenamento dos casos de testes.

A figura 6.3 abaixo exibe a tela do sistema responsável por efetuar a importação de casos de testes.

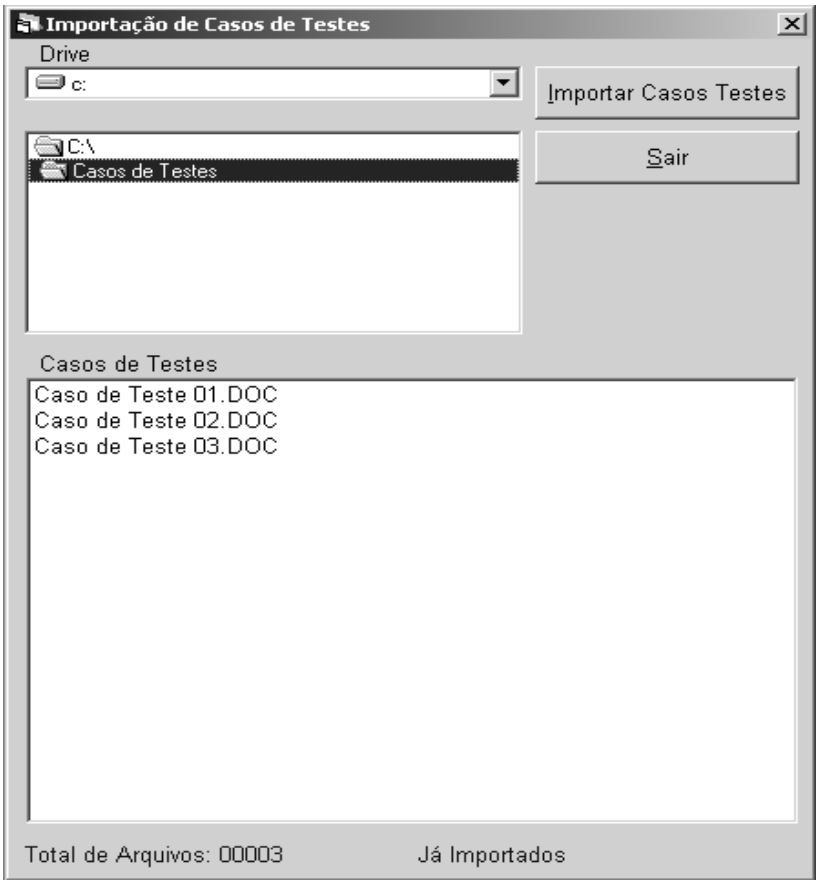


Figura 6.3 Importação de casos de testes

6.2.2 Importação de componentes

A importação de componentes consiste em especificar o caminho físico do servidor de aplicações onde residem os componentes a importar e, por fim, efetuar a importação. São importados para o banco de dados, o nome do componente e seu tipo, “ASP”, “COBOL”, etc. Uma particularidade importante da importação consiste na análise de dependência realizada em cada componente pertencente ao servidor de aplicações. Através da análise do código fonte de cada componente, são encontrados também, os componentes que são utilizados por cada um deles. Com isso, são importados todos os componentes do servidor de aplicações, mais aqueles encontrados pela análise de dependência de código. Um exemplo disso são programas COBOL chamados pelas páginas Web, os quais residem no *mainframe* e não no servidor de aplicações. Essa situação do exemplo torna-se útil, no caso específico de se por acaso, um programa COBOL for alterado, e seja necessário testar novamente todas as páginas Web que o acessam. A figura 6.4 abaixo exibe a tela do sistema responsável por efetuar a importação de componentes.

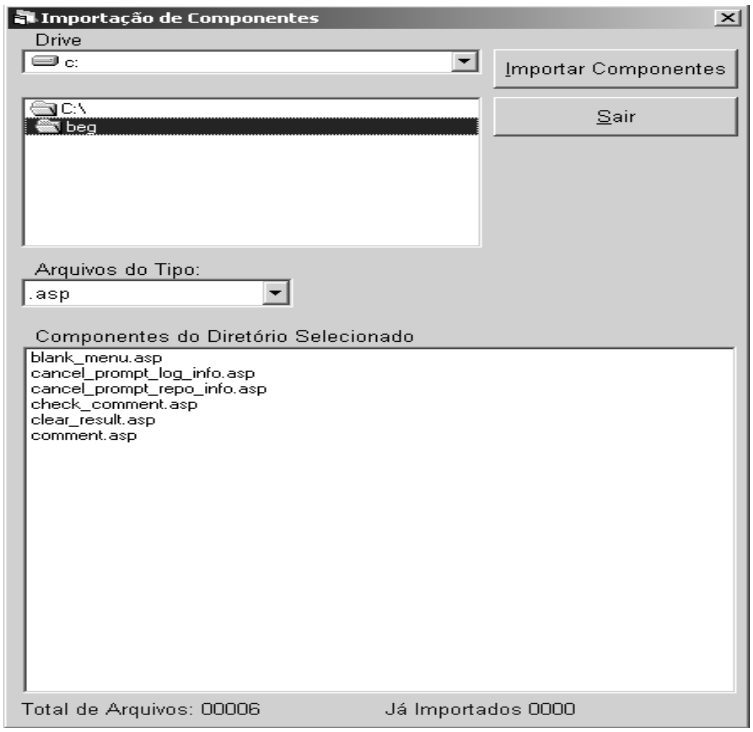


Figura 6.4 Importação de componentes

6.2.3 Importação de casos de uso

A importação de casos de uso está integrada com a ferramenta da *Rational*, "*Rational Rose Enterprise Edition*". Esta integração visa a importação dos casos de uso diretamente a partir dos modelos *Rose* para, posteriormente, em um segundo passo, poderem ser armazenados no banco de dados da ferramenta. Com isso, evita-se a digitação de todos os casos de uso, além do que, consegue-se a sincronização com os modelos desenvolvidos pela equipe de desenvolvedores. Ou seja, conforme os casos de uso são gerados pela equipe de desenvolvimento, os mesmos já poderão ir sendo importados para a ferramenta de testes.

A figura 6.5 abaixo exibe a tela do sistema responsável por efetuar a importação de casos de uso.

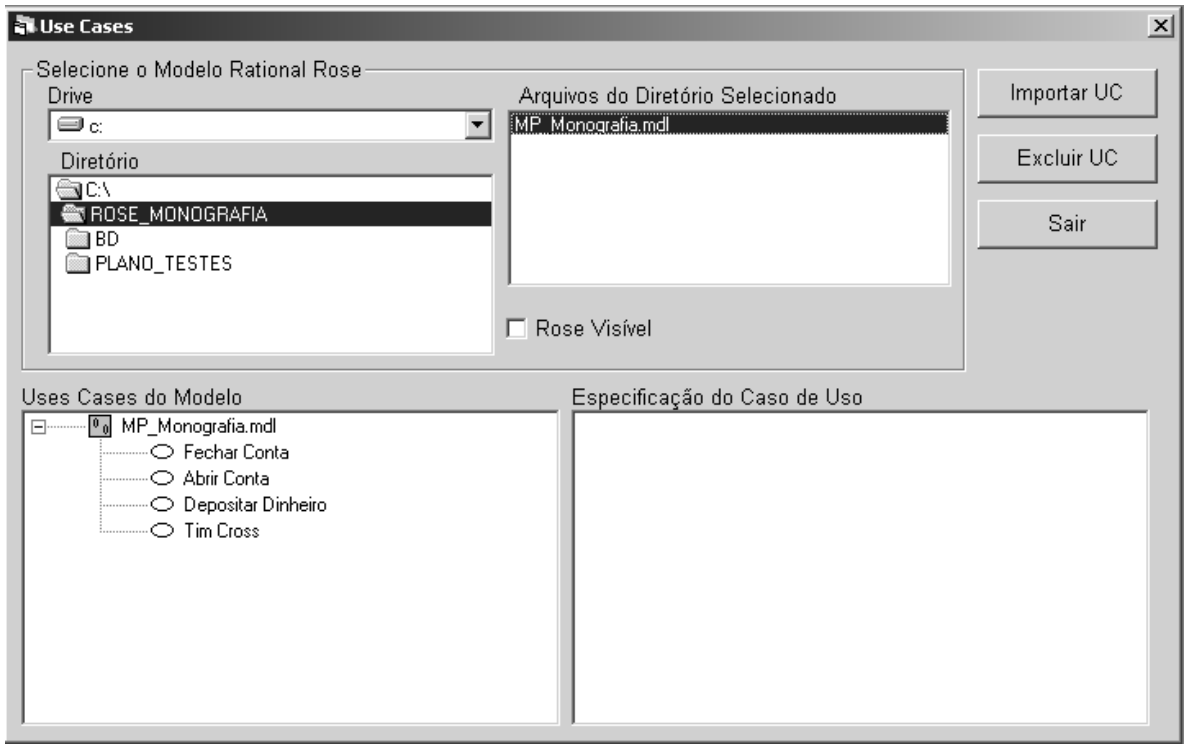


Figura 6.5 Importação de casos de uso oriundos de um modelo Rose

6.2.4 Definindo as rastreabilidades (casos de uso x componentes)

Consiste em especificar: o modelo Rose, o caso de uso e o componente a serem associados. Com isso, vai sendo montada a rastreabilidade entre casos de uso e

componentes, ou seja, quais componentes "*implementam*" quais casos de uso.

A figura 6.6 exibe a tela do sistema responsável por efetuar essa associação.

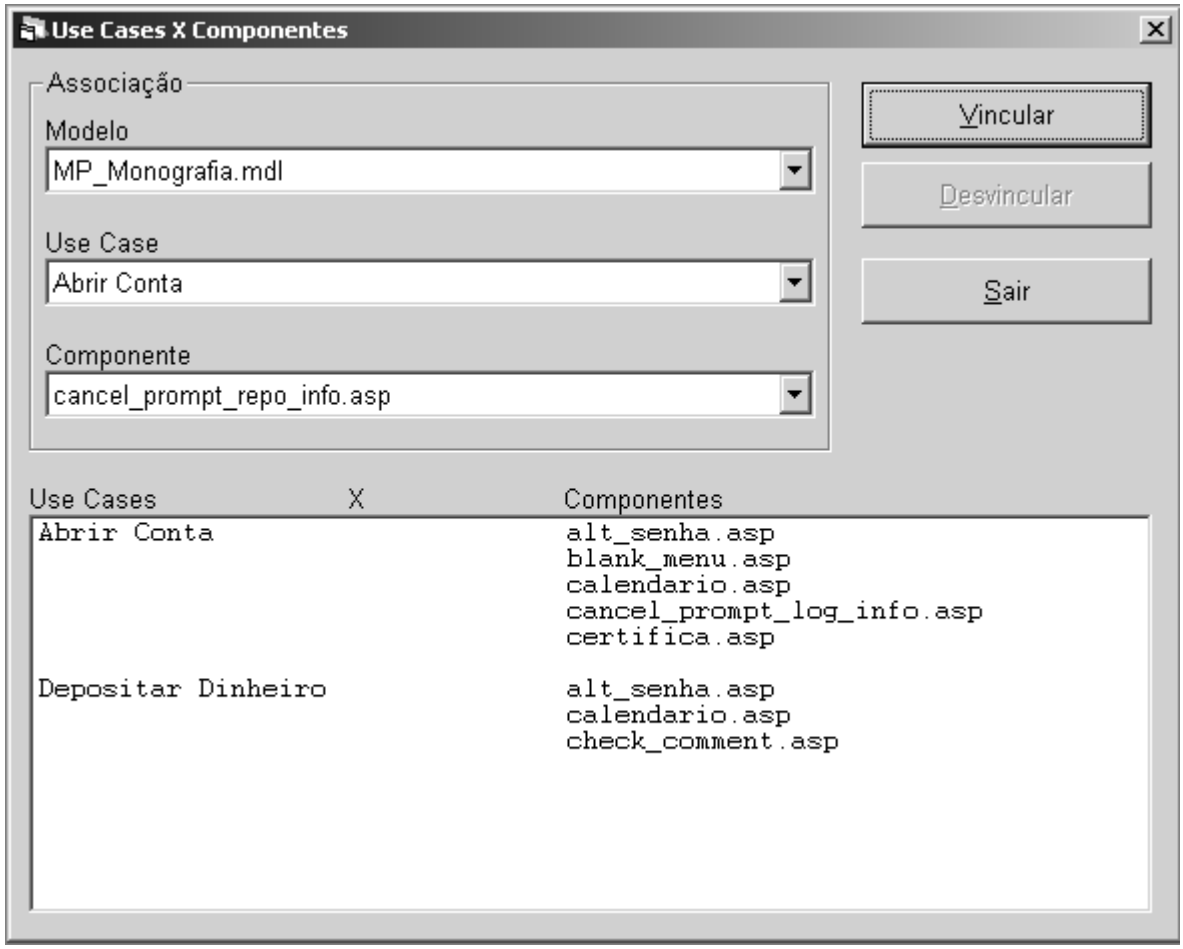


Figura 6.6 Definindo a rastreabilidade entre casos de uso e componentes

6.2.5 Definindo as rastreabilidades (casos de uso x casos de testes)

Consiste em especificar o modelo Rose, o caso de uso e o caso de testes a serem associados. Com isso, vai sendo montada a rastreabilidade entre casos de uso e casos de testes.

A figura 6.7 exibe a tela do sistema responsável por efetuar essa associação.

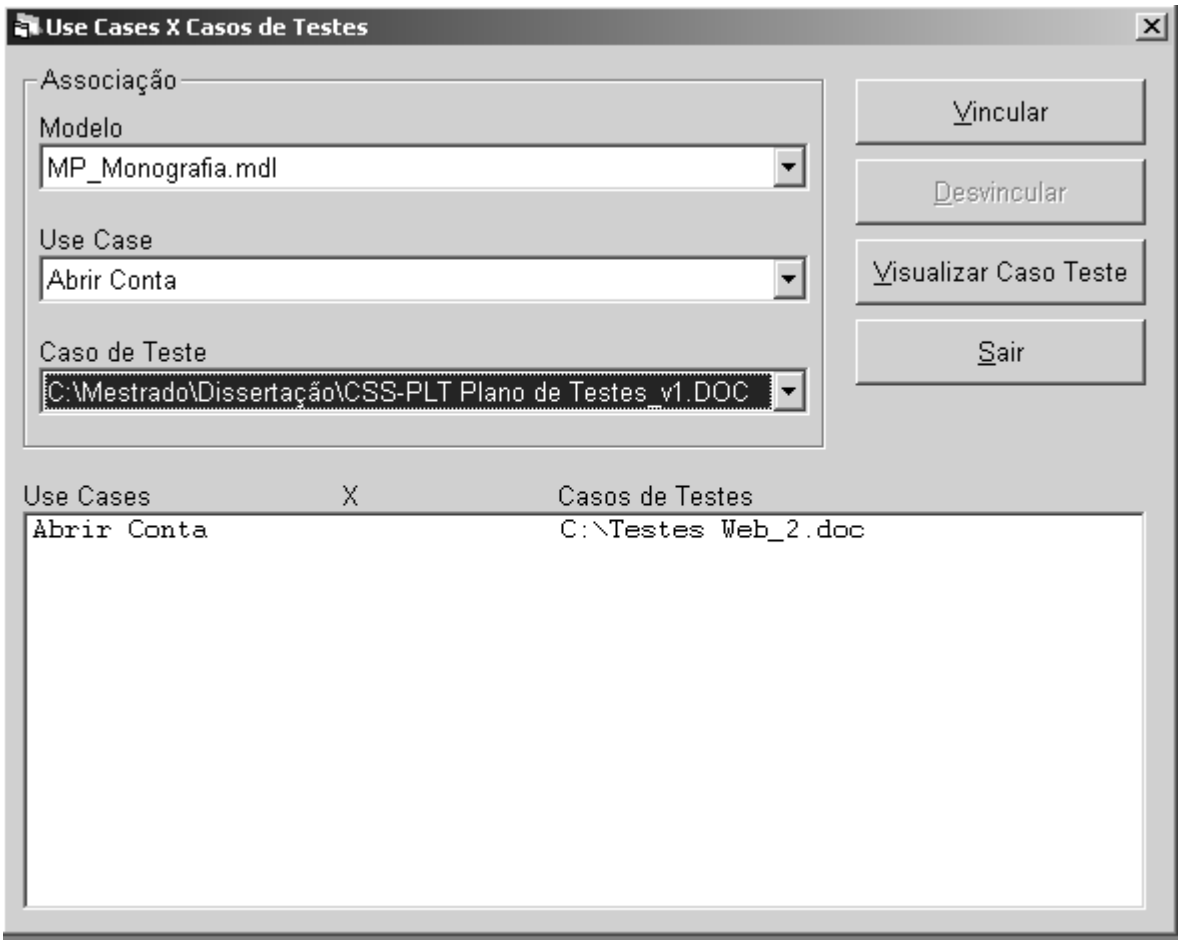


Figura 6.7 Definindo a rastreabilidade entre casos de uso e casos de testes

6.2.6 Listas (casos de testes, componentes e casos de uso)

Consiste na visualização do estado atual do banco em relação aos cadastros de: casos de testes, componentes e casos de uso. As figuras 6.8, 6.9 e 6.10 exibem, respectivamente, as consultas de “casos de uso X casos de testes”, “componentes”, e “casos de uso”.

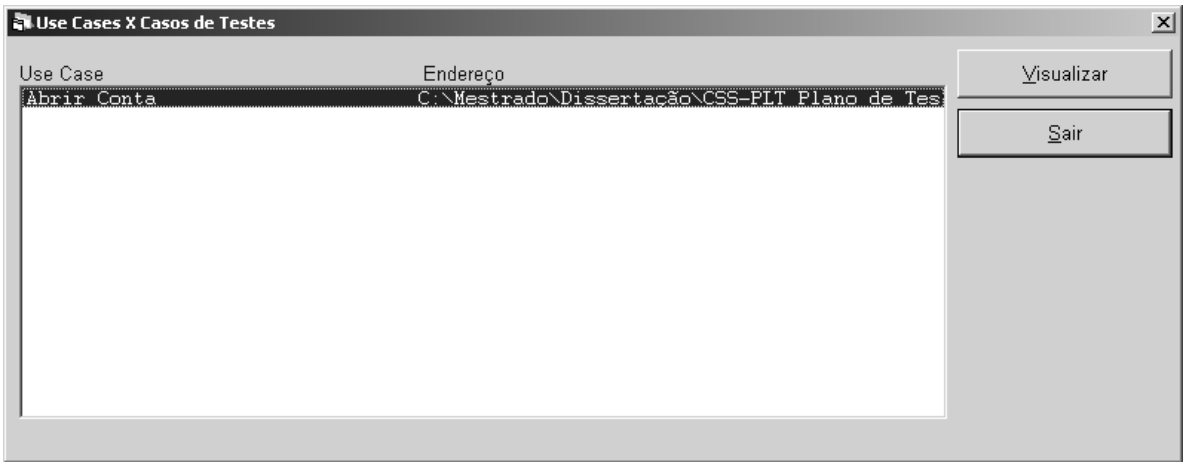


Figura 6.8 Consulta de associações entre casos de uso e casos de testes

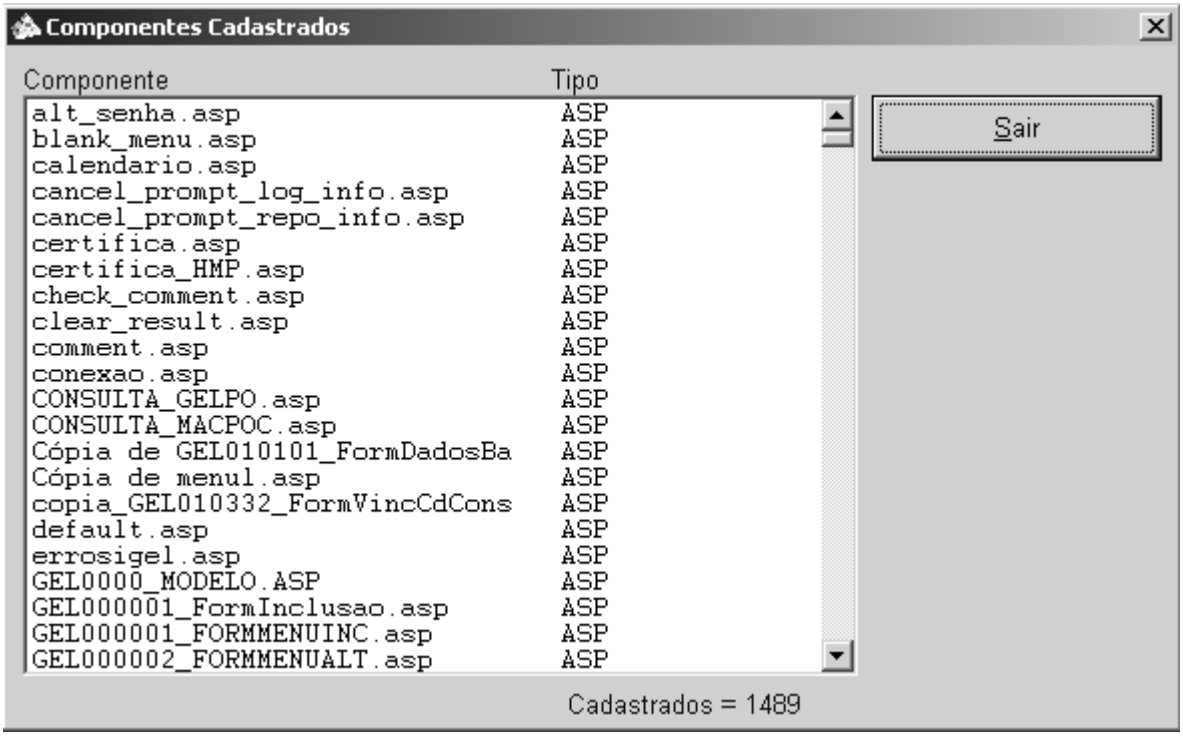


Figura 6.9 Consulta de componentes cadastrados

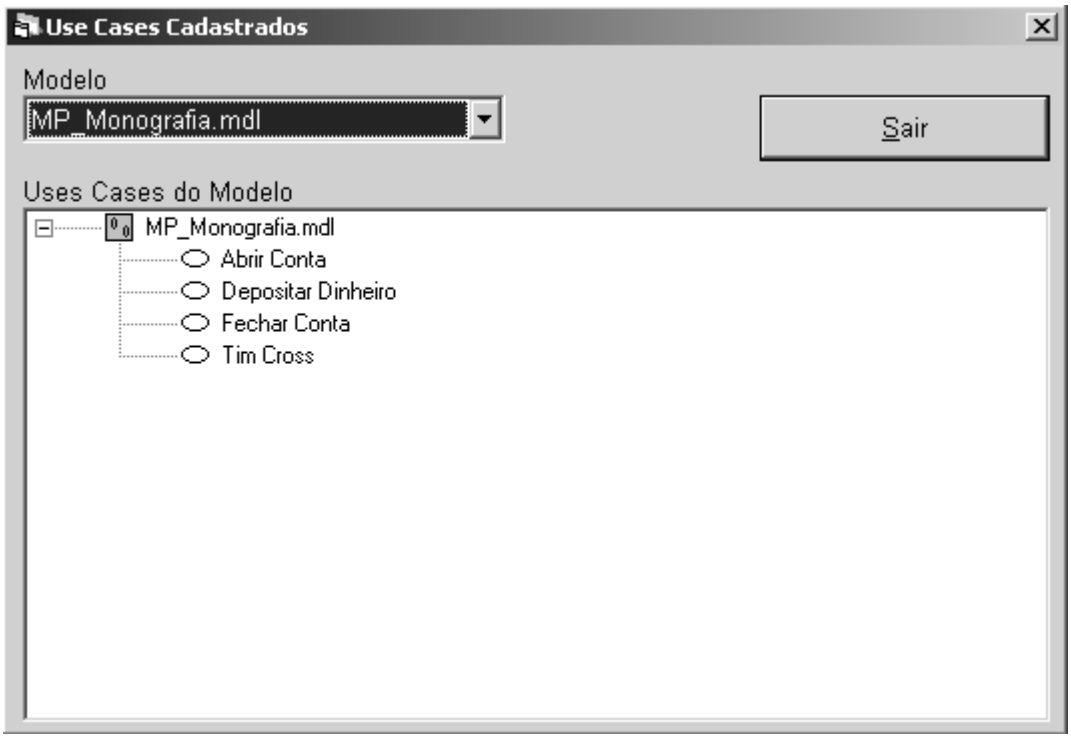


Figura 6.10 Consulta de casos de uso cadastrados

6.2.7 Encontrando o Firewall

Para encontrar o Firewall, principal objetivo da ferramenta, dada uma modificação efetuada no software, deverão ser executados os seguintes passos:

- definir o servidor de aplicações onde se encontram as páginas Web;
- informar o componente modificado;
- pressionar o botão "Encontrar *Firewall*". Na figura 6.11, o botão já aparece com o nome de "Limpar", por já ter sido efetuado o processo de encontrar o firewall. O processo de encontrar o *Firewall* consiste na análise do código fonte dos componentes do servidor de aplicações, verificando se nos mesmos há referências (dependências) em relação ao componente modificado.

Ao final da execução dos passos acima, é encontrado então, o firewall formado pelos componentes que "*usam"* e *"são usados"* pelo componente modificado informado. P*ara* cada componente contido no firewall, são listados os respectivos casos de uso e

casos de testes. Os casos de testes podem ser encontrados a partir dos casos de uso associados aos componentes, ou diretamente, partindo do próprio componente, sem ter que “passar” primeiro pela associação entre componentes e casos de uso.

A figura 6.11 exibe a tela do sistema, em que é realizada a obtenção do Firewall.

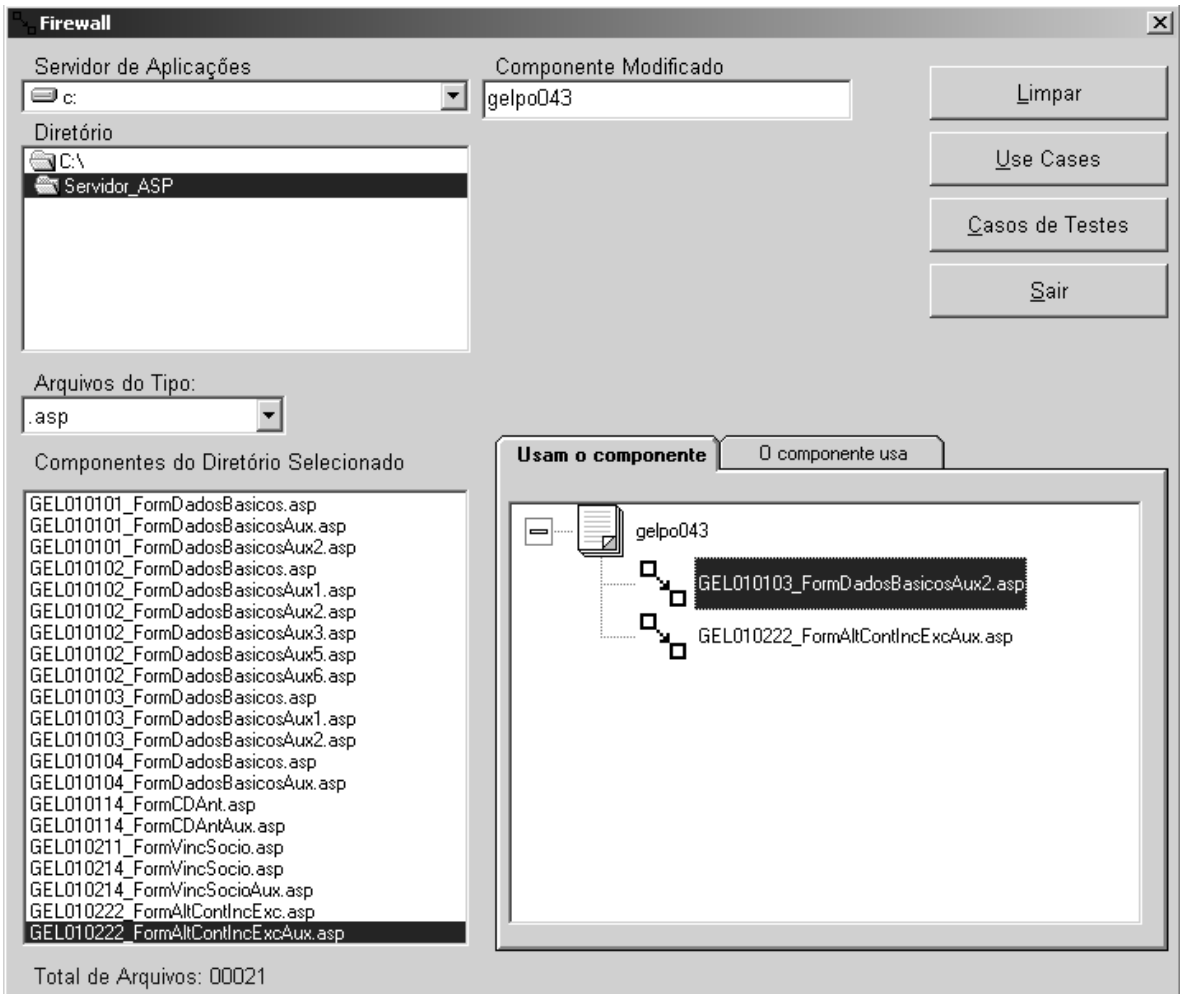


Figura 6.11 Obtenção do firewall

No exemplo apresentado na figura acima, foi informado como modificado o componente “GELPO043” (programa COBOL on-line). A ferramenta analisou o código fonte de cada página Web contida no servidor de aplicações especificado, e encontrou 02 páginas Web que têm dependência em relação ao programa COBOL informado.

Na figura 6.12 seguinte, já são exibidos os casos de uso associados a um dos componentes encontrados dentro do firewall, “*GEL10103_Form_DadosBasicos.asp*”.

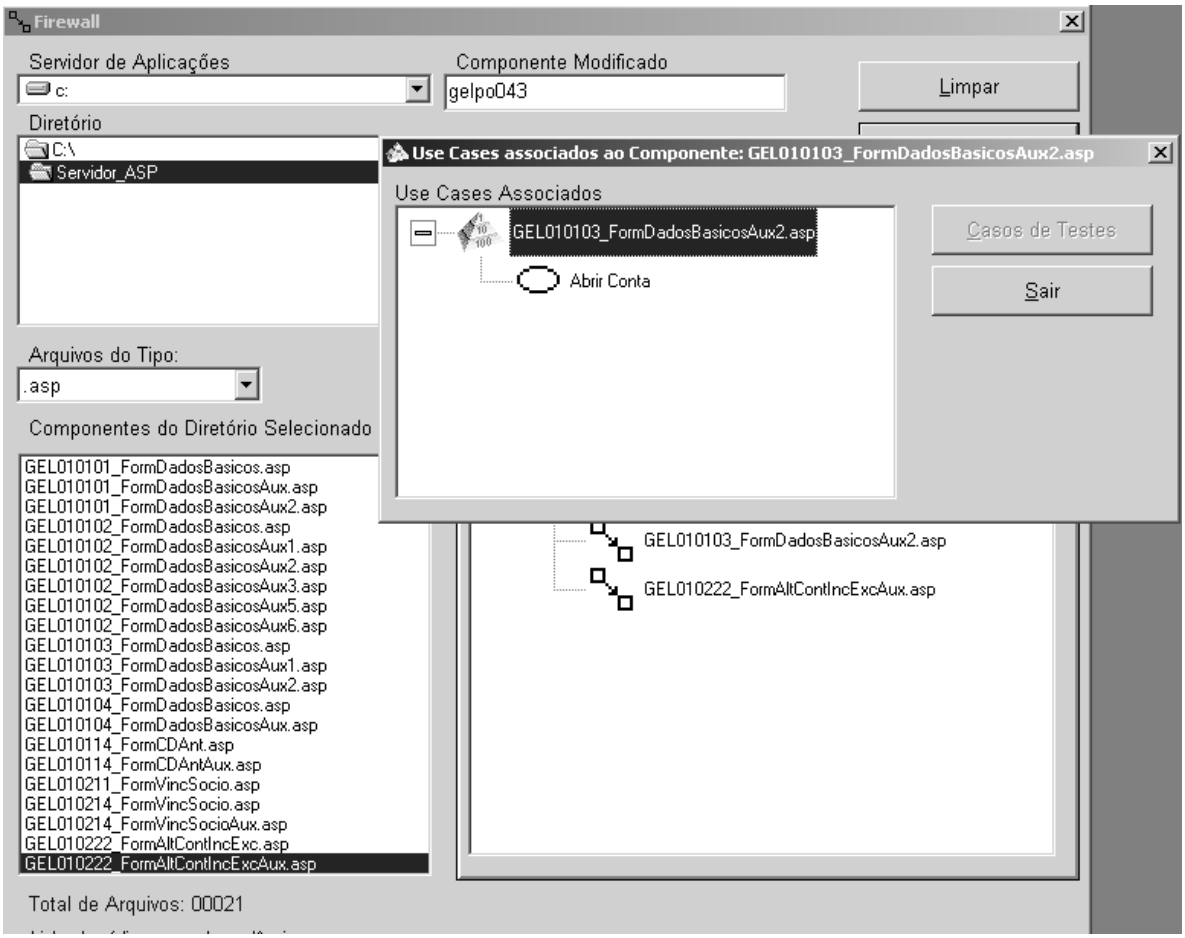


Figura 6.12 Casos de uso associados ao componente dentro do firewall

Dados os casos de uso associados aos componentes, pode-se então chegar aos respectivos casos de testes. A figura 6.13 seguinte exibe tal processo.

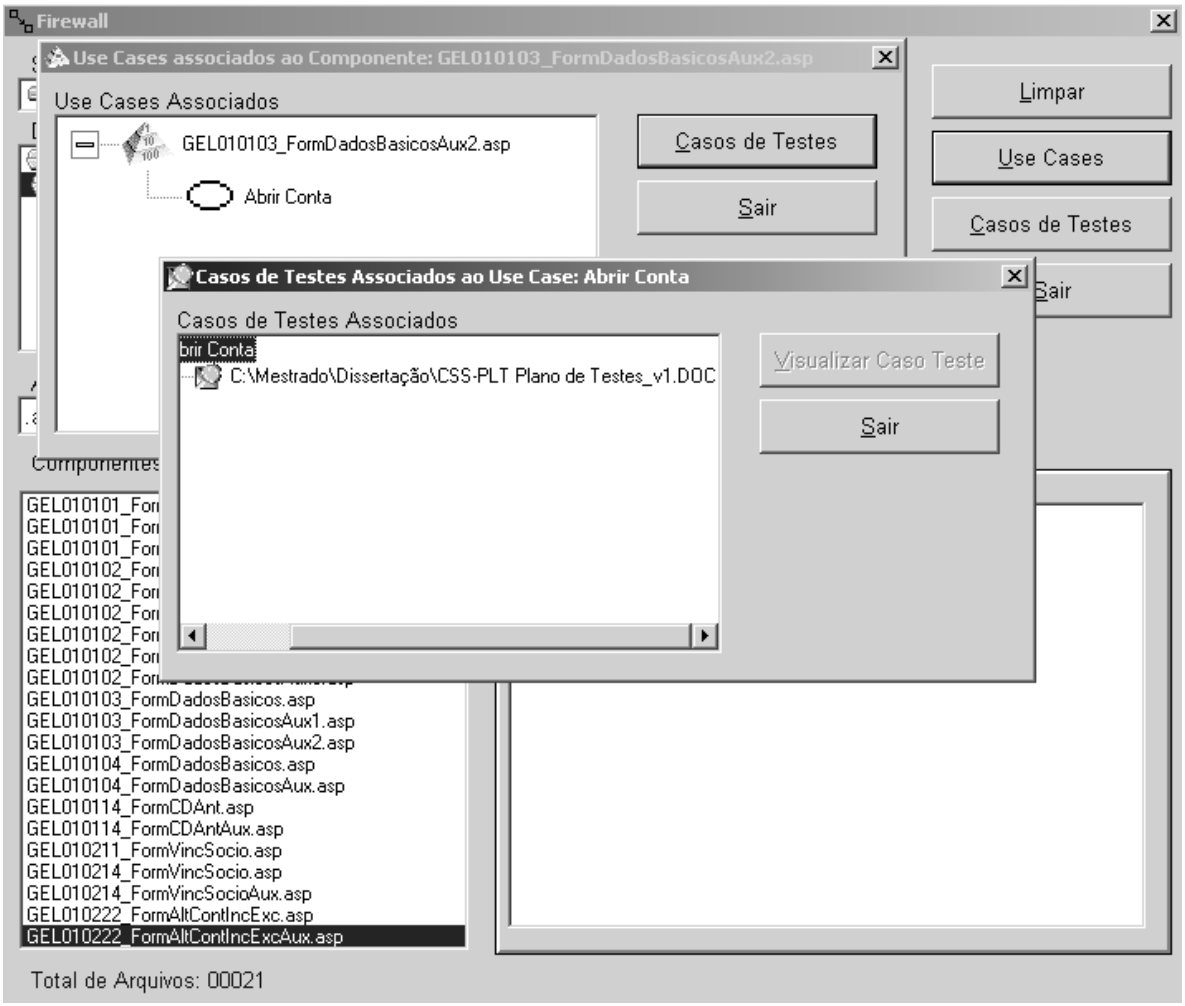


Figura 6.13 Casos de testes associados ao use case dentro do firewall

6.3 Refinamento dos Casos de Testes

Uma vez encontrados os casos de testes relacionados ao firewall, deverá ser executada então a terceira fase da estratégia, que é a de “Refinamento dos Casos de Testes”. Nesse ponto deverão ser observados, por exemplo, se os valores de entrada expressos nos casos de testes ainda são compatíveis com os respectivos componentes.

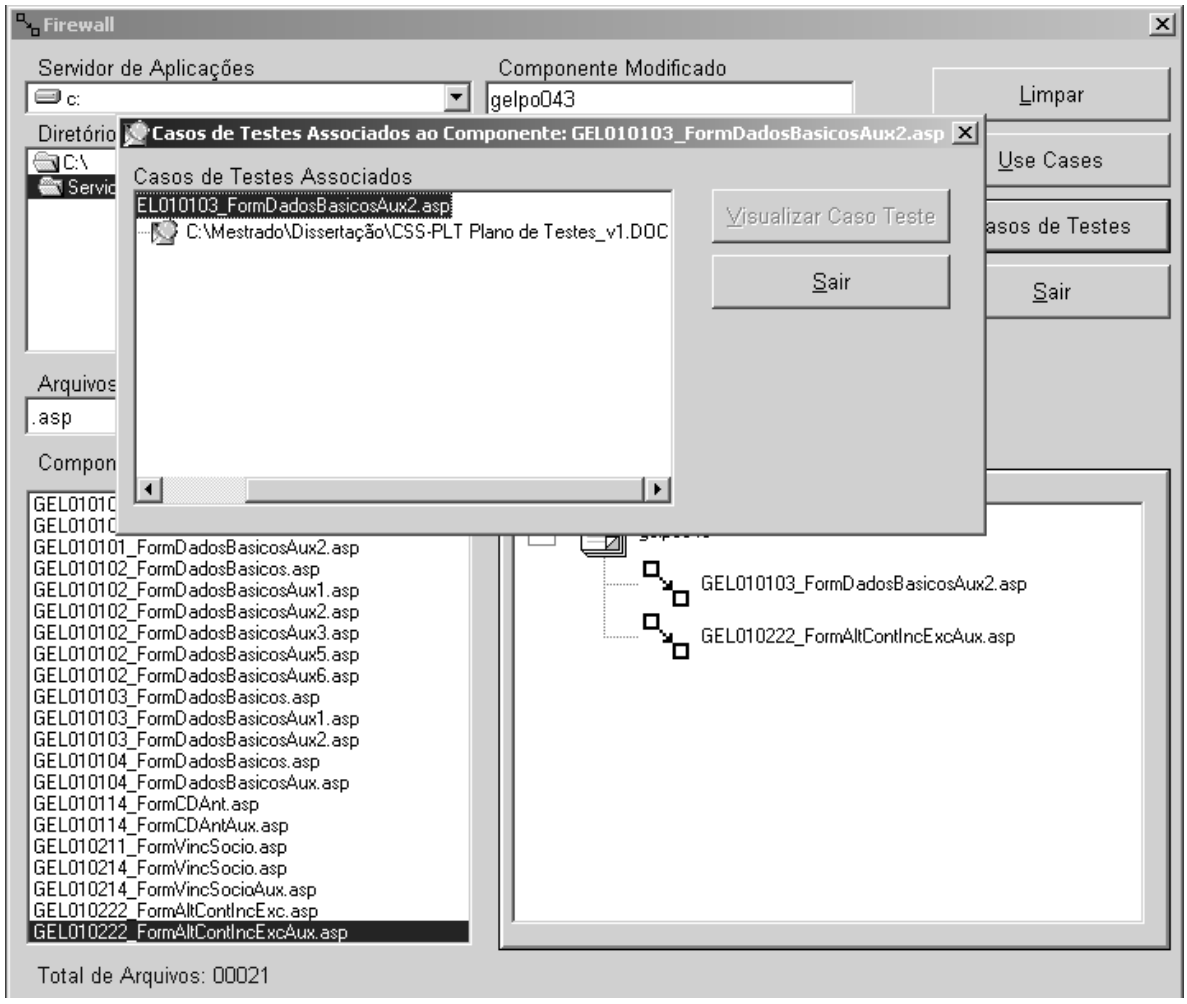


Figura 6.14 Casos de testes associados ao componente dentro do firewall

A outra forma seria chegar aos casos de testes diretamente a partir do componente, como é mostrado na figura 6.14, acima.

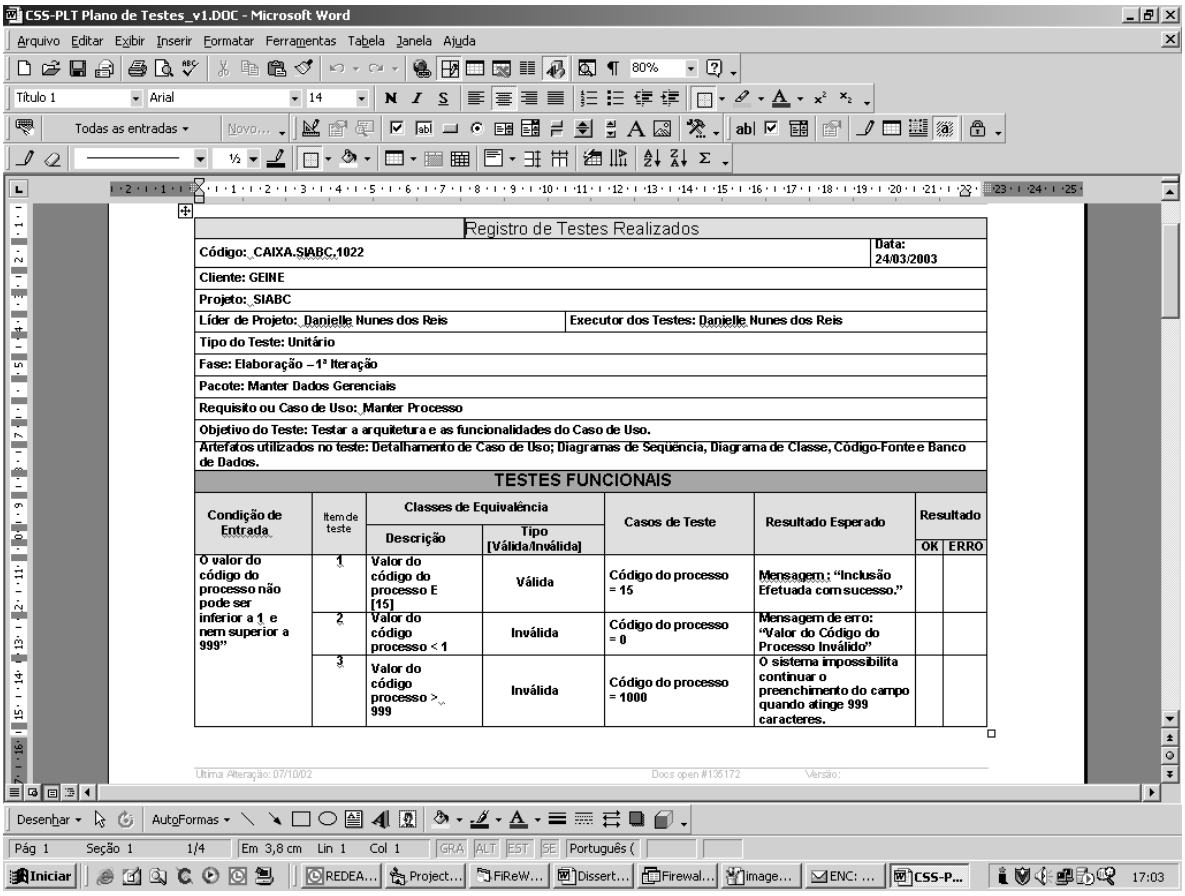


Figura 6.15 Casos de testes visualizado no MS Word a partir da ferramenta

Como a ferramenta está integrada com o MS Word, os casos de testes podem ser visualizados diretamente a partir da ferramenta, como é mostrado na figura 6.15, acima.

6.4 Limitações da Ferramenta

Atualmente a ferramenta está preparada para analisar dependências de código apenas entre: páginas web desenvolvidas em ASP, programas COBOL e arquivos "INCLUDE" (arquivos que possuem procedimentos e funções) que são utilizados pelas páginas Web. Isso devido ao fato do ambiente onde a mesma ter sido testada (Área de Desenvolvimento de Sistemas da CAIXA ECONÔMICA FEDERAL), tal tecnologia, no presente, ser a predominante entre as aplicações Web.

6.5 Considerações Finais

No capítulo foram apresentadas as funcionalidades da ferramenta *FireWeb* associadas a cada fase da estratégia proposta, bem como as limitações constantes nessa primeira versão da ferramenta.

Vimos que a ferramenta *FireWeb* conseguiu dar suporte a duas das três fases da estratégia proposta, apresentada no capítulo anterior, que são: “Preparação da Infra-estrutura” e “Obtenção do Firewall”, e assim, espera-se reduzir o esforço gasto com a atividade de testes de regressão de aplicações Web. A terceira fase da estratégia, “Refinamento dos casos de testes”, por enquanto, ainda não é tratada pela ferramenta.

Capítulo 7 - CONCLUSÃO E TRABALHOS FUTUROS

Ressaltou-se nesse trabalho a importância das atividades de testes na fase de manutenção de software, principalmente pelo fato dessa fase ser responsável por cerca de 90% de todo o custo envolvido com um sistema.

Outro fator importante estudado foi como a rápida difusão da Internet e as novas tecnologias estão produzindo um significativo crescimento da demanda de aplicações Web, sendo que cada vez mais são exigidos requisitos de usabilidade, confiança, interoperabilidade e segurança.

Vimos também que as atividades de testes voltadas a aplicações Web encontram-se ainda muito imaturas e que muito ainda há por ser feito nessa área [GIU02].

7.1 Síntese e Contribuições do trabalho

Foram então apresentados os principais conceitos de aplicações Web, particularidades dos testes para esse tipo de aplicação e, mais especificamente, testes de regressão.

Também foram apresentadas as técnicas de testes existentes a serem aplicadas durante os testes de regressão, que são “Retestar tudo”, “Retestar casos de uso mais críticos”, “Retestar por perfil”, “Retestar código modificado” e “Retestar dentro do *firewall*”. Para cada técnica, foram apresentadas suas principais idéias e uma avaliação, segundo os critérios sugeridos de inclusão, precisão, eficiência, generalidade e suporte à cobertura.

Formulação de uma estratégia a ser empregada durante os testes de regressão de aplicações Web baseada na técnica de “Retestar dentro do Firewall”, a qual foi apresentada no capítulo 5 do presente trabalho.

Visando demonstrar a capacidade de automação da estratégia proposta, e

facilitar o entendimento da mesma, foi construída e apresentada no capítulo 6 a ferramenta "*FireWeb*". Essa ferramenta permitiu automatizar muitos dos passos da estratégia proposta, o que pode tornar o processo de aplicação dos testes de regressão de aplicações Web mais rápido e barato.

7.2 Trabalhos Futuros

Para ilustrar possíveis trabalhos futuros, podemos citar:

- ampliar a ferramenta *FireWeb* para tratar dependências de código de outras linguagens, como Java, Perl, etc;
- adicionar na ferramenta a capacidade de reconhecer o tipo de alteração realizada nos componentes (implementação ou contrato) e, com isso, poder direcionar o refinamento dos casos de testes após ser encontrado o conjunto de testes pertencente ao firewall;
- armazenar na ferramenta os dados de entrada do conjunto de casos de testes existente e, com base no refinamento de testes necessário apontado pela técnica, atualizar os dados de entrada dos casos de testes de forma automática;
- realizar avaliações do uso da técnica em aplicações Web de tamanhos consideráveis, visando avaliá-la de forma mais precisa e, com isso, poder ajustá-la a possíveis problemas que ocorram durante o efetivo uso.
- Integrar a estratégia proposta nesse trabalho a um processo de desenvolvimento, como, por exemplo, o "Processo Unificado", o qual hoje vem se tornando uma tendência de mercado.

REFERÊNCIAS

[BIN99] Binder, Robert; **Testing OO Systems: Models, Patterns and Tools**; Addison-Wesley, 1999, cap. 08, 14, 15.

[BOJ00] Bojic, Dragan; Dugan Velasevic; **A Use Case Method of Architecture Recovery for Program Understanding and Reuse Engineering**; University of Belgrade, Faculty of Electrical Engineering; Bulevar revolucije 73, 11000 Belgrade, Yugoslavia; 2000.

[BOO98] G. Booch, J. Rumbaugh, And I. Jacobson. **The Unified Modeling Language – User Guide**. Addison-Wesley; 1998.

[DIA01] Diane Stottlemyer; **Automated Web Testing Toolkit**; Wiley Computer Publishing; 2001.

[FIL01] Filippo Ricca and Paolo Tonella; **Analysis and Testing of Web Applications**; Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)

[GIU02] Giuseppe Antonio Di Lucca, Anna Rita Fasolino, Francesco Faralli, Ugo De Carlini; **Testing Web Applications**; Proceedings of the International Conference on Software Maintenance (ICSM'02)

[GRA96] Graves, Todd L. et al. **An Empirical study of regression test selection techniques**. ACM Transactions on Software Engineering and Methodology, 15p., 1996

[HAR94] M.J Harrold and Gregg Hothermel; **A Framework for Evaluating Regresion Test Selection Techiniques**; Proc. of the 16th Int'l Conf. On Softw. Eng.; Sorrento, Italy, May 1994, pages 201-210.

[HAR96] M.J Harrold and Gregg Hothermel; **Analyzing Regression Test Selection Techniques**; IEEE Transactions on Software Engineering. V22, nº8, August 1996, pages 529-551.

[HAR97] M.J Harrold and Gregg Hothermel; **A Safe, Efficient Regression Test Selection Technique**; ACM Transactions on Software Engineering and Methodology, V.6. no. 2, April 1997, pages 173-210.

[HUN01] Hung Q. Nguyen; **Testing Applications on the Web**; Wiley Computer Publishing; 2001.

[JAC03] site pesquisado em 08/11/2003
http://www.dsc.ufpb.br/~jacques/cursos/apoo/html/proj1/proj2.htm

[LIU00] Chien-Hung Liu, David C. Kung, Pei Hsia, Chih-Tung Hsu; **Structural Testing of Web Applications**; Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE'00)

[MAR00] Martins, Eliane.; **Manutenção e Ferramentas CASE**. IC-UNICAMP, 2000.

[MAR01] site pesquisado em 30/08/2003
http://www.dcc.unicamp.br/~eliane/Cursos/Transparencias/VVTestes/tregr.pdf

[MYE79] G.J.Myers. **The Art of Software Testing**. John-Wiley & Sons, 1979.

[PIG97] Pigoski, Thomas M; **Practical Software Maintenance: Best Practices for Managing Your Software Investiment**. Wiley Computer Publishing, 1997.

[PRE02] Pressman, Roger S.; **Engenharia de Software** , 5. Edição. McGraw-Hill, Inc., 2002.

[PRE95] Pressman, Roger S.; **Engenharia de Software**, 3. Edição. McGraw-Hill, Inc., 1995.

[RIO03] Rios, Emerson e Moreira Filho, Trayahu; **Projeto & Engenharia de Software – Teste de Software**, ed. Alta Books LTDA, 2003.

[SET89] Setzer, Valdemar W.; **Bancos de Dados – Conceitos, Modelos, Gerenciadores, Projeto Lógico, Projeto Físico**, ed. Edgar Blücher LTDA, 1989.

[VOL01] Volpi, Lisiane Maes.; Dissertação de Mestrado – **Uma Estratégia de testes de software para Ambiente Cliente Servidor**, Setor de Ciências Exatas – UFPR, 2001.

APÊNDICE A - FERRAMENTAS PARA AUTOMAÇÃO DE TESTES DE APLICAÇÕES WEB

Ferramentas para Testes de Carga

Ferramenta	Descrição	Requisitos	URL
WebSpray	Utilizada para testes de carga, inclui capacidades para testes de links, pode simular até 1000 clientes a partir de um único endereço IP, também suporta múltiplos endereços IP.	Win98/2000 e NT 4.0	www.rehillnetworks.com
TestWorks/Web	Coleção de ferramentas para testes Web para gravação e reprodução. Também faz teste de carga.	Win95/NT e Unix	www.soft.com/products/web/index.html
WebPerformance Trainer	Suporta até 200 usuários por gravação a várias velocidades de conexão. Suporta todos os browsers e servidores Web.	WinNT, Linux, Solarix, e demais variantes Unix	www.webperfcenter.com

Ferramenta	Descrição	Requisitos	URL
WebSizr/WebCorder	Teste de carga e ferramentas para gravação e reprodução. Suporta autenticação, cookies e redirecionamentos.	Win95 ou WinNT. Como é baseada em Java, requer JDK 1.0.2 ou posterior.	www.technovations.com
Benchmark Factory	Testes de carga para Ecommerce para soluções Cliente-Servidor. Inclui gravação e reprodução. Possui capacidade de comparação de performance entre bancos de dados distintos como: MS SQL Server, Oracle 7 e 8, Sybase 11, ODBC, DB2 CLI e Informix.	WinNT	www.benchmarkfactory.com

Ferramenta	Descrição	Requisitos	URL
MS Web Application Stress	Testes de stress da Microsoft. Possui capacidades para gravação e reprodução. Suporta ASP e cookies.	Para testes de páginas Web construídas com ASP rodando em NT Server ou Win2000, MSIE 4.0 ou posterior.	www.homer.rte.microsoft.com
FORECAST	Testes de carga para softwares para web Facilita, cliente-servidor, redes e sistemas de bancos de dados	Plataformas Unix	www.facilita.co.uk/
Zeus Free Web Load Test Tool	Ferramenta de uso livre de testes de carga	Plataformas Unix	www.webper.zeus.co.uk/intro.html
VeloMeter	Ferramenta para testes web baseada em Java. Inclui código fonte.	Requer JDK 1.1.6 ou posterior	www.binevolve.com
http-Load	Ferramenta Free para testes de Carga.	Unix	www.acme.com/software/http_load
Microsoft WCAT	Testes de Carga para aplicações sobre MS Internet Information Server IIS.	WinNT	www.msdn.microsoft.com/workshop/server/toolbox/wcat.asp

Ferramenta	Descrição	Requisitos	URL
Portent Web Load Test Tool	Testes de Carga	Escrita em Java, multiplataformas	www.loadtesting.com
WebArt	Testes de Carga, simula até 200 usuários. Também inclui testes funcionais e de regressão baseados em gravação e reprodução.	Win3.1/95/ NT	www.oclc.org/webart
WebLoad	Ferramenta de testes de carga da Computer Associates CA.	Win95/NT, Solaris e AIX.	www.ca.com/products/platinum/appdev/fe_iltps.htm/
Radviews WebLoad	Ferramenta de testes de carga da Radview Software. Suporta sessões, cookies, proxies, autenticação, HTML dinâmico e múltiplas plataformas.	Win95/NT, Solaris e AIX	www.radview.com

Ferramentas para Testes de Aplicações Java

Ferramenta	Descrição	Requisitos	URL
Panorama for Java	Contém 06 ferramentas Java integradas. JavaSQA para avaliar a qualidade de softwares	Win95/95/ NT	www.softwareautomation.com/

Ferramenta	Descrição	Requisitos	URL
	orientados a objetos. JavaDocGen para análise estática de código. JavaStructure para geração de diagramas e analisar a estrutura do código. JavaDiagrammer para análise lógica do código Java, análise de fluxo de controle e diagramação. JavaTest para análise de cobertura de testes e minimização de casos de testes. JavaPlayback para operações GUI, realiza gravação e reprodução de movimentos.		
McCabe Visual Test	Possui capacidades para avaliar a cobertura e outras métricas.	Java	www.mccabe.com
JTest!	Ferramenta de testes caixa branca	Java	www.parasoft.com/index.html
AppletLoad	Parte do conjunto de ferramentas Radview WebLoad, para testes de performance de	Radview's WebLoad	www.radview.com

Ferramenta	Descrição	Requisitos	URL
	Applets implementados em Java		

Ferramentas para Verificação de Links

Ferramenta	Descrição	Requisitos	URL
LinkGuard Online	Software livre na Web, possui serviços para checagem de links quebrados	Internet	www.linkguard.com
Xenu's Link Sleuth	Software livre, suporta SSL (Secure Sockets Layer) Web sites	Win95/NT	http://home.snafu.de/tilman/xenulink.html
Linkalarm	Software livre por algum período. Programação de execução automática e relatórios por e-mail.	Internet	www.linkalarm.com/
Theseus	Verificador de links para Mac; versão de avaliação disponível.	Macintosh	www.matterform.com/
Alert Linkrunner	Verificador de links	Win95/NT	www.altertbook-marks.com/lr/download.com
I-Control WebWeaver	Verificador de links	Win95/NT	www.futurearts.com/
RiadaLinx	Verificador de Links, cópia de avaliação disponível	Win95/NT	www.riada.com
Linkbot	Verificador de links,	Win95/NT	www.tetranet-

Ferramenta	Descrição	Requisitos	URL
	páginas faltando, páginas lentas, etc.		software.com/products/linkbot.html
InfoLink	Verificador de Links. Versões freeware e para avaliação disponíveis.	Win95	www.biggbyte.com/

Validadores de HTML

Ferramenta	Descrição	Requisitos	URL
RealValidator	Validador de HTML Shaware. Suporta várias versões de HTML.	Win95/98/ NT; MSIE 4.0 ou posterior, HTML Help 1.1	http://arealvalidator.com/
CSE 3310 HTML Validator	Validador de HTML shaware.	Win95/NT	www.htmlvalidator.com/

Validadores de HTML e Verificadores de Links de uso livre

Ferramenta	Descrição	Requisitos	URL
WDG HTML Validator	Suporta última versão de HTML, mensagens de erro amigáveis.	Web site	www.htmlhelp.com/tools/validator
MetaMedic	Versão profissional disponível	Web site	http://northern.webs.com/set/setsimjr.html
Web Page Purifier	Suporta HTML 2.0, HTML 3.2, HTML 4.0	Web site	www.delorie.com/web/purify.html

Ferramenta	Descrição	Requisitos	URL
	ou WebTV 1.1		
W3C HTML Validation Service	Trata uma URL por vez.	Web site	http://validator.w3.org/
NetMechanic	Digite no site a URL a ser checada e depois receba via e-mail o resultado.	MSIE ou Netscape	www.netmechanic.com
Bobby	Trata AOL browsers, MSIE, Netscape	Web site	www.cast.org/bobby/
Doctor HTML	Analisa por página ou por site.	Web site	www2.imagiware.com/RxHTML/index_noframes.html
EWS Weblint Gateway	Site com validador HTML on-line.	Web site	www.cen.uiuc.edu/cgi-bin/weblint
Web Page Backward Compatibility Viewer	Site com validador HTML on-line.	Web site	www.delorie.com/web/wpbcv.html
Mindlt	Envia e-mail se uma URL é atualizada. Pode ser usado para manter o controle de mudanças externas de links a documentos e páginas web	Web site	www.netmind.com/html/individual.html

Ferramentas para Análise de Log

Ferramenta	Descrição	Requisitos	URL
HTTPD Log Analyzers list	Mais extensiva ferramenta de listagem de logs na Web.	Unix, WinNT, Mac	www.uu.se/Software/Analyzers/Access-analyzers.html
Web Developers' Virtual Library Log Analyzer Listing	Inclui cerca de 30 ferramentas com descrições	Web site	www.stars.com/Vlib/Software/Statistics.html

Ferramentas para Testes Funcionais e de Regressão

Ferramenta	Descrição	Requisitos	URL
Macrobot	Capacidades de gravação e reprodução	NT4.0, SP4 ou posterior, Win95/98 MSIE 4.0 SP1 ou posterior	www.watchfire.com

Ferramentas para testes de segurança

Ferramenta	Descrição	Requisitos	URL
HackerShield	Examina servidores, estações, hubs, etc.	NT 4.0 com SP 4, MSIE	www.bindview.com/
Cyber Attack Defense System	Monitora e analista atividades suspeitas	Solaris e WinNT	www.checkpoint.com

Ferramenta	Descrição	Requisitos	URL
ITS4	Software de código aberto. Procura em códigos C e C++ por potenciais vulnerabilidades de segurança.	Unix e Windows com CygWin.	www.rstcorp.com/its4
Inspectorscan	Análise e relato de problemas de segurança	WinNT	www.shavlik.com