UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

CAMPUS LUIZ MENEGHEL

LUCAS FERRARI

FERRAMENTAS DE TESTE PARA APLICAÇÕES WEB: UM ESTUDO DE CASO

Bandeirantes

2008

LUCAS FERRARI

FERRAMENTAS DE TESTE PARA APLICAÇÕES WEB: UM ESTUDO DE CASO

Trabalho de Conclusão de Curso submetido ao Campus Luiz Meneghel da Universidade Estadual do Norte do Paraná, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. José Reinaldo Merlin

Bandeirantes

2008

LUCAS FERRARI

FERRAMENTAS DE TESTE PARA APLICAÇÕES WEB: UM ESTUDO DE CASO

Trabalho de Conclusão de Curso submetido ao Campus Luiz Meneghel da Universidade Estadual do Norte do Paraná, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____

Prof. José Reinaldo Merlin
Universidade Estadual do Norte do Paraná – Campus Luiz Meneghel

_____

Prof. Viviane de Fátima Bartholo
Universidade Estadual do Norte do Paraná – Campus Luiz Meneghel

_____

Prof. Cristiane Yanase Hirabara de Castro
Universidade Estadual do Norte do Paraná – Campus Luiz Meneghel

Bandeirantes, ___ de_____de 2008

Aos meus familiares, amigos e namorada pela compreensão, paciência, apoio durante todos esses anos de esforço. Vocês são meu maior incentivo.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais Irineu e Mariângela, aos irmãos e também às pessoas mais próximas, que estiveram sempre me apoiando em todos os momentos, me incentivando, e compreendendo minhas necessidades para a conclusão desse trabalho.

À Deus, que sempre presente em minha vida, me proporcionou saúde, sabedoria e disposição para enfrentar as dificuldades, crendo que eu fosse capaz de superá-las.

Agradeço aos meus amigos, companheiros de sala, e todos que sempre estiveram presentes em minhas atividades, que compartilharam dos momentos de tristezas e alegrias ao meu lado.

Aos professores, que estiveram prontamente atentos a sanar minhas dúvidas e me serviram com bases para evoluir no desenvolvimento deste trabalho.

Ao orientador, Professor José Reinaldo Merlin, que me apoiou a todos os instantes, pacientemente, e dedicadamente para a conclusão desse trabalho.

E a todos que direta ou indiretamente auxiliaram na realização deste.

“Nossas dúvidas são traidoras e nos fazem perder o que, com freqüência, poderíamos ganhar, por simples medo de arriscar.”

William Shakespeare

FERRARI, Lucas; **Ferramentas de Teste para Aplicações Web: Um Estudo de Caso.** Novembro 2008. 50f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Universidade Estadual Norte do Paraná, campus Luiz Meneguel, Bandeirantes, 2008.

RESUMO

O uso de aplicações web é cada dia mais comum no mercado de softwares, e paralelo a essa ascensão, surge a necessidade de que essas aplicações sejam de qualidade, seguras e eficientes. Para atingir o nível de qualidade desejado, é necessário a realização de testes dessas aplicações web e, com o intuito de agilizar esse processo de testes, foram desenvolvidas ferramentas automatizadas para garantir a produtividade, eficiência e eficácia nos testes. O presente trabalho visa estudar ferramentas de teste de aplicações web. Deste modo, foram escolhidas a ferramenta Selenium IDE para a realização de testes funcionais e a ferramenta Apache JMeter para efetuar testes de desempenho. Nesta monografia foi realizado então um estudo de caso para analisar a viabilidade da utilização dessas ferramentas de teste automatizadas no processo de teste de aplicações web.

Palavras-chave: teste de aplicações web, ferramentas de teste automatizadas, produtividade, eficiência, eficácia.

FERRARI, Lucas; **Test Tools for Web Applications: a Case Study.** November 2008. 50f. Course Completion Work (Bachelor of Information System) – Universidade Estadual Norte do Paraná, campus Luiz Meneguel, Bandeirantes, 2008.

ABSTRACT

The use of web applications each day is more common in the market of software, and parallel to this ascension, it is necessary that these applications are high-quality, secure and efficient. To achieve the desired level of quality, it is necessary testing of these web applications and with a view to speed the process of testing, automated tools were developed to ensure the productivity, efficiency and effectiveness in the tests. This paper aims to study tools for testing web applications. Thus, were chosen the Selenium IDE tool to make functional tests and Apache JMeter tool to perform performance tests. This monograph was then a case study to examine the feasibility of using these tools to test automated the process of testing web applications.

Keywords: web applications test, automated testing tools, productivity, efficiency, effectiveness.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto

A Engenharia de Software é uma disciplina que atua diretamente em todos os processos de desenvolvimento de software, pois, segundo Pressman (2002), ela visa estabelecer técnicas, critérios, métodos e ferramentas para a produção de software de alta qualidade, alta produtividade e baixo custo. A fim de garantir essa qualidade do produto final e visando diminuir a ocorrência de falhas, é fundamental a utilização de uma das atividades de Garantia de Qualidade que a Engenharia de Software dispõe, o teste.

A atividade de teste tem influência significativa no resultado final do software, e devido à sua importância e seu alto custo de realização, técnicas e critérios de teste têm sido desenvolvidos a fim de tornar essa atividade mais eficiente.

Em relação ao desenvolvimento de software, além da criação de aplicações convencionais, atualmente com a popularização da Internet, é cada vez mais crescente o desenvolvimento de aplicações Web. Essas aplicações têm a mesma estrutura de aplicações convencionais, contudo, cada vez mais são exigidos requisitos de usabilidade, confiança, interoperabilidade e segurança, pois trabalham com uma alta carga de informações, num espaço de tempo muito rápido, e com uma grande heterogeneidade de tecnologias para o seu desenvolvimento. Decorrente dessas características, é que os testes de aplicações Web necessitam de maior atenção, embora o desenvolvimento de aplicações Web tenha algumas diferenças em relação ao desenvolvimento de aplicações desktop, algumas das diretivas de teste destas aplicações podem ser implantadas em aplicações Web.

No entanto, a utilização de testes em uma aplicação pode ser comprometida se o teste não for executado de maneira correta e eficiente. O teste efetuado de forma manual pode não trazer resultados consistentes, pois etapas podem deixar de ser executadas, dependendo do tamanho da aplicação. Como auxílio para esse problema, foram desenvolvidas ferramentas para execução de testes automatizados, que geram casos de teste, trabalhando como um usuário virtual, varrendo toda a estrutura da aplicação em busca de falhas, que venham a comprometer a qualidade e o funcionamento do software.

Contudo, mesmo sabendo que não há um processo geral para a execução de testes que confirmem a corretitude da aplicação, o teste contribui significativamente para aumentar a confiança de que o software desempenha as funções para as quais foi desenvolvido.

1.2 Objetivos

1.2.1 Objetivo Geral

Este trabalho tem o objetivo de realizar um estudo de caso sobre ferramentas de teste para aplicações Web, em que ferramentas de teste serão selecionadas e aplicadas no teste de aplicações Web, analisando-se a eficácia e eficiência na detecção de defeitos. Com isto pretende-se analisar vantagens, desvantagens e dificuldades no uso de ferramentas de teste automatizadas.

1.2.2 Objetivos Específicos

- Utilizar-se dos conhecimentos adquiridos durante o curso relacionados com a disciplina de Engenharia de Software;
- Ressaltar as diferenças entre uma aplicação Web e uma aplicação *desktop*, no que se refere ao teste de software;
- Levantar as ferramentas de teste de aplicações Web mais conhecidas;
- Selecionar ferramentas, aplicá-las no teste de aplicações web e analisar os resultados.

1.3 Justificativa

A crescente demanda por software baseado na Web vêm impondo mudanças no processo de desenvolvimento. As aplicações precisam ser desenvolvidas cada vez mais rapidamente, estão sujeitas a mudanças constantes e requerem um alto nível de qualidade (PRESSMAN, 2002). Defeitos que poderiam ser tolerados em uma aplicação *desktop* são inaceitáveis em uma aplicação Web. Por exemplo, um *site* de comércio eletrônico com problemas de desempenho pode fazer o consumidor

desistir da compra, causando prejuízos à empresa e insatisfação do cliente (MYERS, 2004).

No aspecto científico, a utilização de ferramentas de teste demonstra as melhorias que a evolução tecnológica pode oferecer para agilizar o processo de testes, aumentando a eficácia e eficiência desta atividade. Em relação à contribuição social, esse projeto visa melhorar a qualidade das aplicações Web, para que seus usuários não sejam atingidos diretamente por defeitos inconvenientes em suas aplicações.

Em resumo, este trabalho justifica-se pela importância do teste de software na elaboração de produtos confiáveis e pela necessidade de se utilizar ferramentas para aumentar a produtividade e eficiência dos testes.

1.4 Metodologia

Este trabalho pode ser classificado como uma pesquisa exploratória do tipo m j estudo de caso (GIL, 2008).

Quanto à natureza, é um trabalho teórico-empírico. A revisão bibliográfica é utilizada para conhecer sobre teste de software, aplicações Web e teste de aplicações Web. Também serão levantadas as ferramentas para teste Web mais conhecidas.

Quanto aos objetivos, é uma pesquisa exploratória, pois visa proporcionar ao pesquisador maior familiaridade com o fenômeno ou problema.

Quanto ao tipo, é uma pesquisa baseada em estudo de caso, pois consiste no estudo detalhado, em profundidade, de poucos objetos de estudo.

Os materiais necessários à instrumentação da pesquisa são: um microcomputador, uma ferramenta de teste e uma *IDE – Integrated Development Enviroment* – para executar a aplicação, além da própria aplicação objeto de estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a revisão da literatura sobre os três principais assuntos tratados: Teste de Software, Teste de Aplicações Web e Ferramentas de Teste de Aplicações Web.

2.1 Teste de Software

A Engenharia de Software evoluiu significativamente nas últimas décadas procurando estabelecer técnicas, critérios, métodos e ferramentas para a produção de software, em conseqüência da crescente utilização de sistemas baseados em computador em praticamente todas as áreas da atividade humana. Isto provoca uma crescente demanda por qualidade e produtividade, tanto do ponto de vista do processo de produção como do ponto de vista dos produtos gerados. A Engenharia de Software pode ser definida como uma disciplina que aplica os princípios de engenharia com o objetivo de produzir software de alta qualidade a baixo custo (PRESSMAN, 2002).

O processo de desenvolvimento de software envolve uma série de atividades nas quais, apesar das técnicas, métodos e ferramentas empregados, erros no produto ainda podem ocorrer. Atividades agregadas sob o nome de Garantia de Qualidade de Software têm sido introduzidas ao longo de todo o processo de desenvolvimento, entre elas atividades de VV&T – Verificação, Validação e Teste, com o objetivo de minimizar a ocorrência de erros e riscos associados. Dentre as técnicas de verificação e validação, a atividade de teste é uma das mais utilizadas, constituindo-se em um dos elementos para fornecer evidências da confiabilidade do software em complemento a outras atividades, como por exemplo, o uso de revisões e de técnicas formais e rigorosas de especificação e de verificação (MALDONADO, 1991).

A atividade de teste consiste de uma análise dinâmica do produto e é uma atividade relevante para a identificação e eliminação de erros que persistem. Ainda, o conjunto de informação oriundo da atividade de teste é significativo para as atividades de depuração, manutenção e estimativa de confiabilidade de software (PRESSMAN, 2002). Salienta-se que a atividade de teste tem sido apontada como uma das mais onerosas no desenvolvimento de software (PRESSMAN, 2002).

Apesar deste fato, Myers (2004) observa que aparentemente conhece-se muito menos sobre teste do que sobre outros aspectos e/ou atividades do desenvolvimento de software.

Teste de software é um elemento crítico de garantia de qualidade e representa a última revisão da especificação, projeto e geração de código. Um ponto crucial na atividade de teste, independentemente da fase, é o projeto e/ou a avaliação da qualidade de um determinado conjunto de casos de teste T utilizado para o teste de um produto, dado que, em geral, é impraticável utilizar todo o domínio de dados de entrada para avaliar os aspectos funcionais e operacionais de um produto em teste. O objetivo é utilizarem-se casos de teste que tenham alta probabilidade de encontrar a maioria dos defeitos com um mínimo de tempo e esforço, por questões de produtividade. Segundo Myers (2004), o principal objetivo do teste de software é revelar a presença de erros no produto. Portanto, o teste bem sucedido é aquele que consegue determinar casos de teste para os quais o programa em teste falhe. Essa atividade pode ser vista como uma atividade destrutiva (PRESSMAN, 2002).

O teste de produtos de software envolve basicamente quatro etapas: planejamento de testes, projeto de casos de teste, execução e avaliação dos resultados dos testes (BEIZER, 1990). Essas atividades devem ser desenvolvidas ao longo do próprio processo de desenvolvimento de software, e em geral, concretizam-se em três fases de teste: de unidade, de integração e de sistema.

2.1.1 Fases de Teste de Software

Além da utilização de técnicas e critérios de teste, quando grandes programas são testados é necessário dividir a atividade de testes em várias fases. Com isso o testador pode se concentrar em aspectos diferentes do software e utilizar diferentes estratégias de seleção de dados de teste e medidas de cobertura em cada uma delas. De maneira geral, a atividade de teste pode ser considerada como uma atividade incremental realizada em três fases, tais como: teste de unidade, teste de integração e teste de sistema.

2.1.1.1 Teste de Unidade

Os testes de unidade focalizam cada unidade para assegurar que aspectos do algoritmo de cada uma delas esteja corretamente implementado. O objetivo é identificar falhas relacionadas à lógica e implementação em cada unidade. Durante essa fase utiliza-se muito a técnica de teste estrutural que requer a execução de elementos específicos da estrutura de controle de cada unidade, com o objetivo de garantir uma completa cobertura e máxima detecção de erros.

Nesta fase é comum a necessidade de desenvolver *drivers* e *stubs.* Seja F a unidade a ser testada, o *driver* é responsável por coordenar o teste de F. Ele coleta os dados fornecidos pelo testador, passa para F na forma de argumentos, recebe os resultados produzidos por F e os apresenta ao testador. Um *stub* é uma unidade que substitui, durante a fase de teste, uma outra unidade usada (chamada) por F. Geralmente, um *stub* é uma unidade que simula o comportamento de uma unidade usada com um mínimo de esforço de computação ou manipulação de dados. O desenvolvimento de *drivers* e *stubs* pode representar um grande esforço adicional no teste de unidade (VINCENZI *et al*, 2003).

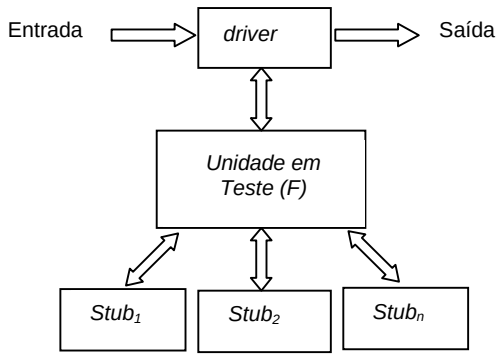

Figura 1 – Representação de *drivers* e *stubs* utilizados no teste de unidade

2.1.1.2 Teste de Integração

Uma vez testados os componentes individuais do programa, eles devem ser integrados para criar um sistema parcial ou completo. Esse processo de integração envolve construir e testar o sistema resultante quanto a problemas que surjam a partir das interações de componentes. Os testes de integração devem ser desenvolvidos a partir da especificação do sistema e começar assim que as versões

de alguns dos componentes do sistema estejam disponíveis (SOMMERVILLE, 2003).

Os testes de integração são necessários porque unidades que individualmente trabalham corretamente podem não atuar da mesma forma quando em conjunto. Isto decorre das limitações do teste de unidade, que não garante que cada unidade trabalhe adequadamente em todas as situações.

A principal dificuldade que surge nos testes de integração é localizar erros descobertos durante o processo. Há complexas interações entre os componentes de um sistema e, quando uma saída anômala é descoberta, pode ser difícil encontrar a origem do erro. Para facilitar a localização de erros, deve-se sempre utilizar uma abordagem adicional para a integração e os testes do sistema. Inicialmente, é necessário integrar uma configuração mínima do sistema e testar esse sistema. Em seguida, são adicionados componentes a essa configuração mínima, que é testada depois de cada componente adicionado (SOMMERVILLE, 2003).

2.1.1.3 Teste de Sistema

Após a realização dos testes de integração, o software é combinado com outros elementos como hardware e bancos de dados, por exemplo. Através do teste de sistema é possível verificar se todos esses elementos combinam-se adequadamente e se o desempenho global requerido é atingido pelo sistema.

Segundo Pressman (2002), software é apenas um dos elementos de um sistema baseado em computador. O software é incorporado a outros elementos do sistema e são necessários testes para essa integração. Tais testes saem do escopo do processo de software e não são conduzidos apenas por engenheiros de software, necessitando da participação de administradores de rede, administradores de banco de dados, etc.

Teste de sistema é, na verdade, uma série de diferentes testes cujo principal propósito é exercitar completamente o sistema baseado em computador. Embora cada teste tenha diferentes propósitos, todos objetivam verificar que os elementos do sistema foram integrados apropriadamente e desempenharam as funções a eles alocadas.

A seguir estão apresentados alguns dos testes de sistemas mais comuns.

-Teste de Segurança: alguns sistemas lidam com informações valiosas ou vitais. Em tais sistemas uma penetração ilegal pode causar prejuízos (ou benefícios indevidos) aos indivíduos. Invasões podem ser feitas com diferentes intenções. *Hackers* podem invadir por esporte, empregados ou ex-empregados podem invadir por vingança, golpistas podem invadir com intenção de obter vantagens monetárias ilícitas. Testes de segurança objetivam verificar se os mecanismos de proteção presentes no sistema estão, de fato impedindo o acesso não autorizado.

O teste de segurança tem seu foco em duas áreas principais de segurança:

- *Segurança em nível de aplicação*: assegura que, baseados na segurança desejada, atores são restringidos a funções específicas ou casos de uso, ou são limitados aos dados que são disponibilizados a eles de acordo com o perfil definido.

- *Segurança em nível de sistema*: assegura que apenas aqueles usuários com permissão de acesso são capazes de acessar o sistema. Caso o sistema esteja integrado com outros sistemas, realiza-se o teste de integração.

Esse teste utiliza como critério a criação de testes para cada tipo de usuário e a verificação de cada permissão criando transações específicas para cada um dos tipos. O teste modifica o tipo de usuário e executa novamente os testes para os mesmos usuários. Em cada caso, verificam-se as funções adicionais ou se os dados estão corretamente disponíveis ou negados (DATASUS, 2008).

-Teste de Performance: para sistemas de tempo real ou embutido, é inaceitável que o desempenho do software não esteja de acordo com os requisitos de performance. O teste de performance é projetado para testar o desempenho em tempo de execução no contexto de um sistema integrado. Requer uma instrumentação de hardware e software, pois freqüentemente é necessário medir a utilização de recursos de uma forma exata (por exemplo, ciclos do processador) (PRESSMAN, 2002).

O objetivo é descobrir situações que levam à degradação e possível falha no sistema, além de verificar o comportamento do sistema para funções de transações ou de negócios designadas sob as condições de carga normal de trabalho e de carga limite de trabalho.

Seus critérios vão desde a utilização de procedimentos de teste desenvolvidos para teste funcional ou de ciclo de negócio, quanto à modificação de arquivos de dados para aumentar o número de transações ou os *scripts* para aumentar o número de iterações em que cada transação ocorre (DATASUS, 2008).

- Teste de Carga: o teste de carga submete o sistema à variação de carga de trabalho para medir e avaliar os comportamentos de desempenho e a sua habilidade de continuar funcionando apropriadamente sob cargas de trabalho diferentes. Também avalia as características de desempenho, assim como tempos de resposta, taxas de transações e outros casos sensíveis ao tempo (DATASUS, 2008).

Seus procedimentos são:

- Modificação dos arquivos de dados para aumentar o número de transações ou os testes para aumentar o número de vezes que cada transação ocorre.

- Variação do número de usuários conectados.

-**Teste de recuperação (*recovery*):** muitos sistemas precisam se recuperar a partir de falhas e reassumir o processamento em um tempo pré-determinado, com ou sem intervenção humana. Teste de recuperação é um teste de sistema que força o software a falhar de várias maneiras diferentes e verifica se o sistema se recupera apropriadamente. Neste teste são verificados o tempo de restabelecimento, integridade de dados, recuperação de transações pendentes, etc (PRESSMAN, 2002).

-Teste de *Stress:* são projetados para colocar o sistema em condições não usuais, de forma que sejam exigidos recursos em quantidade, freqüência ou volume anormais. Podem ser exigidos, por exemplo, uma quantidade excessiva de acesso ao banco de dados, um volume de tráfego via rede além da média prevista, um número de usuários simultâneos superior a média esperada.

Os testes de *stress* são particularmente relevantes para sistemas distribuídos, com base em uma rede de processadores. Esses sistemas freqüentemente exibem uma degradação severa quando são sobrecarregados. A rede torna-se inundada com dados de coordenação, que os diferentes processos devem trocar, e, assim, os processos tornam-se cada vez mais lentos, à medida que eles esperam pelos dados solicitados a partir de outros processos (DATASUS, 2008).

Os principais procedimentos utilizados por esse teste são:

- Para teste de recursos limitados, os testes são executados em uma máquina simples, e as memórias RAM e do *drive* de disco no servidor são reduzidos ou limitados.

- Para Testes de Estresse, múltiplos clientes são usados, executando os mesmos testes ou testes complementares, para produzir volumes de transação limite ou de transações mistas (umas de alta e outras de baixo desempenho).

2.1.2 Técnicas e Critérios de Teste

Técnicas e critérios de teste de software fornecem um roteiro para o projeto de testes que exercitem a lógica interna do software e os domínios de entrada e saída para descobrir erros antes da entrega ao cliente (PRESSMAN, 2002). Embora não exista um procedimento de teste de propósito geral que possa ser usado para provar a corretitude de um programa, os testes, se conduzidos sistemática e criteriosamente, contribuem para aumentar a confiança de que o software desempenha as funções especificadas e evidenciar algumas características mínimas do ponto de vista da qualidade do produto. Considerando que os testes exaustivos são impraticáveis, os critérios de teste servem para selecionar e avaliar conjuntos de casos de teste. Desta forma, tais critérios devem fornecer indicação de quais casos de teste devem ser utilizados para aumentar as chances de revelar erros (método de seleção de casos de teste) ou, quando erros não forem revelados, estabelecer um nível elevado de confiança na correção do programa.

2.1.2.1 Teste Funcional

O teste funcional também é conhecido como teste caixa preta (MYERS, 2004) pelo fato de tratar o software como uma caixa cujo conteúdo é desconhecido e da qual só é possível visualizar o lado externo, ou seja, os dados de entrada fornecidos e as respostas produzidas como saída. Na técnica de teste funcional são verificadas as funções do sistema sem se preocupar com os detalhes de implementação. O teste funcional envolve dois passos principais: identificar as funções que o software deve realizar e criar casos de teste capazes de checar se essas funções estão sendo realizadas pelo software (PRESSMAN, 2002). As funções que o software deve possuir são identificadas a partir de sua especificação.

Assim, uma especificação bem elaborada e de acordo com os requisitos do usuário é essencial para esse tipo de teste.

A seguir estão dois dos critérios mais utilizados na técnica funcional: Particionamento em Classes de Equivalência, Análise do Valor Limite.

-***Particionamento em Classes de Equivalência***: a partir das condições de entrada de dados identificadas na especificação, divide-se o domínio de entrada de um programa em classes de equivalência válidas e inválidas. Em seguida seleciona-se o menor número possível de casos de teste, baseando-se na hipótese que um elemento de uma dada classe seria representativo da classe toda, sendo que para cada uma das classes inválidas deve ser gerado um caso de teste distinto. O uso de particionamento permite examinar os requisitos mais sistematicamente e restringir o número de casos de teste existentes. Alguns autores também consideram o domínio de saída do programa para estabelecer as classes de equivalência.

-Análise do Valor Limite: é um complemento ao critério de Particionamento em Classes de Equivalência, sendo que os limites associados às condições de entrada são exercitados de forma mais rigorosa; ao invés de selecionar-se qualquer elemento de uma classe, os casos de teste são escolhidos nas fronteiras das classes, pois nesses pontos se concentra um grande número de erros. O espaço de saída do programa também é particionado e são exigidos casos de teste que produzam resultados nos limites dessas classes de saída.

2.1.2.2 Teste Estrutural

A técnica de teste estrutural apresenta uma série de limitações e desvantagens decorrentes das limitações inerentes às atividades de teste de programa enquanto estratégia de validação (HOWDEN, 1987). Esses aspectos introduzem sérios problemas na automatização do processo de validação de software. Independentemente dessas desvantagens, essa técnica é vista como complementar a técnica funcional e informações obtidas pela aplicação desses critérios têm sido consideradas relevantes para as atividades de manutenção, depuração e confiabilidade de software (PRESSMAN, 2002). Na técnica de teste estrutural, também conhecida como teste caixa branca (em oposição ao nome caixa preta), os aspectos de implementação são fundamentais na escolha dos casos de teste. O teste estrutural baseia-se no conhecimento da estrutura interna da implementação e apoia-se numa representação conhecida como grafo de programa.

Um grafo de programa mostra nós (blocos de instruções executadas sequencialmente) e arestas (desvios de um nó para outro).

Os critérios de teste estrutural são, em geral, classificados em:

-***Critérios Baseados em Fluxo de Controle****:* utilizam apenas características de controle da execução do programa, como comandos ou desvios, para determinar quais estruturas são necessárias. Os critérios mais conhecidos dessa classe são **Todos-Nós** – exige que a execução do programa passe, ao menos uma vez, em cada vértice do grafo de fluxo, ou seja, que cada comando do programa seja executado pelo menos uma vez; **Todos-Arcos** – requer que cada aresta do grafo, ou seja, cada desvio de fluxo de controle do programa, seja exercitada pelo menos uma vez; e **Todos-Caminhos** – requer que todos os caminhos possíveis do programa sejam executados (PRESSMAN, 2002).

Os testes de caminho não testam todas as possíveis combinações de todos os caminhos no programa. O objetivo do teste de caminho é assegurar que cada caminho independente do programa seja executado pelo menos uma vez (SOMMERVILLE, 2003).

-***Critérios Baseados em Fluxo de Dados***: utilizam informações do fluxo de dados do programa para determinar os requisitos de teste. Esses critérios exploram as interações que envolvem definições de variáveis e referências a tais definições para estabelecerem os requisitos de teste (RAPPS & WEYUKER, 1985). Exemplos dessa classe de critérios são os Critérios de Rapps e Weyuker e os Critérios Potenciais-Usos. Os Critérios de Rapps e Weyuker baseiam-se no grafo Def-Uso, que é uma extensão do grafo de programa. Nele são estabelecidas associações entre as definições de variáveis (atribuição de valor) e os usos de tais valores (referências às variáveis). Um dos critérios desse grupo é Todas-Definições, que requer que cada definição de variável seja exercitada pelo menos uma vez. Outro critério, Todos-Usos, requer que todas associações entre uma definição de variável e os subseqüentes usos sejam exercitados através de pelo menos um caminho livre de definição.

2.2 Teste de Aplicações Web

Aplicações web são essencialmente aplicações cliente-servidor, em que o cliente é um navegador web e o servidor é um servidor web ou um servidor de aplicações.

O objetivo do teste de aplicações Web é o mesmo do teste de aplicações convencionais: descobrir erros antes que o software entre em operação. No entanto, a complexidade das aplicações e a interdependência entre os componentes são maiores, o que aumenta a complexidade e a importância deste tipo de teste (MYERS, 2004).

Aplicações Web exigem desenvolvimento rápido e alterações freqüentes, devido ao surgimento de novas tecnologias e oportunidades comerciais, bem como ao *feedback* dos usuários. Por essas razões, os modelos de processo iterativos, baseados em prototipação rápida e mudança contínua, são mais adequados. Ao final de cada iteração, a aplicação deve ser testada (CONALLEN, 2000).

No contexto deste processo iterativo em que são desenvolvidas, o papel da análise é fundamental. Os resultados da análise podem ser utilizados para entender e modificar a aplicação, uma vez que fornecem uma visão de alto nível do sistema existente, mostrando a organização e funcionalidades. Para atingir um alto nível de qualidade, além dos testes funcionais, a aplicação deve ser testada através de critérios estruturais e, para isto, os produtos da análise são utilizados (RICCA & TONELLA, 2001). Por esse motivo a modelagem UML (*Unified Modeling Language*) pode ser utilizada para apoiar o processo de teste, pois ela tem um alto nível de representação que auxilia na avaliação da estrutura estática da aplicação inclusive permitindo a geração de casos de teste para critérios caixa-branca. O teste caixa-branca explora a estrutura interna de uma aplicação Web para definir o critério de cobertura. Portanto, análise é um pré-requisito, pois a partir dela é que são derivados determinados elementos de teste, como por exemplo, fluxos de dados. É ela também o ponto de partida quando os casos de teste são automaticamente ou semi-automaticamente gerados (RICCA & TONELLA, 2001).

2.2.1 Arquitetura Básica de uma Aplicação Web

Aplicações web geralmente são desenvolvidas sob uma arquitetura de três camadas onde cada camada é tratada como uma caixa-preta possuindo interfaces bem definidas. Este modelo de arquitetura permite trabalhar com a estrutura interna de cada camada sem se preocupar em interferir em outra.

A camada 1, que é a camada de apresentação, é responsável pelo conteúdo visual apresentado ao usuário, e é representada pelo servidor web. A camada 2, ou camada de negócios , é utilizada para executar o software que modela os processos de negócio. Algumas de suas funcionalidades são o processamento de transações, autenticação de usuários e validação dos dados.

A camada de dados, ou terceira camada, armazena e recupera dados de um banco de dados, tipicamente um sistema gerenciador de banco de dados (MYERS, 2004).

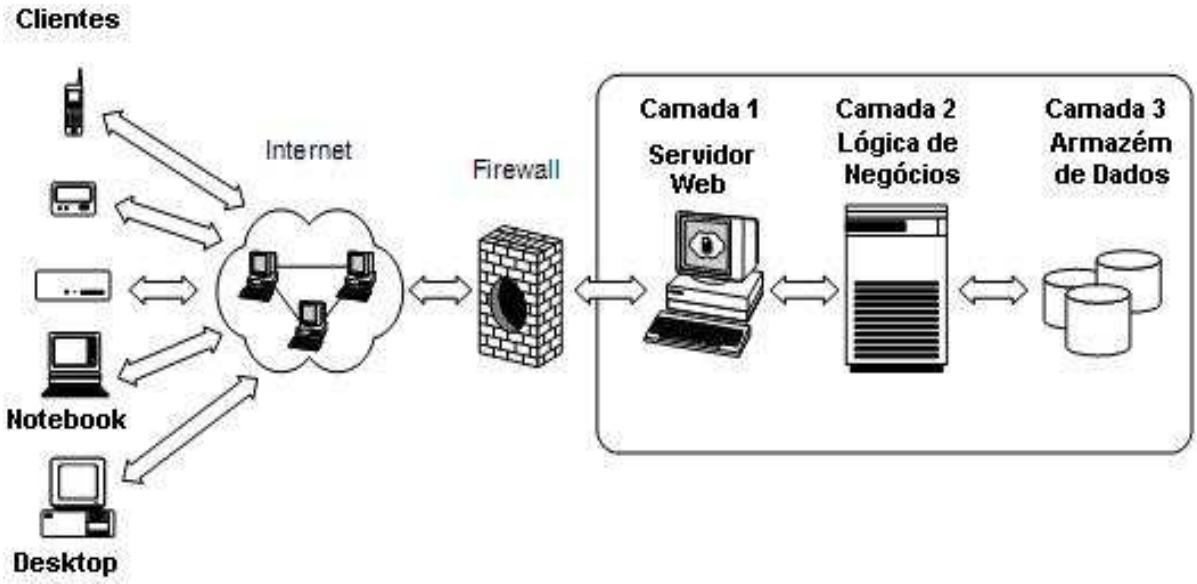

Figura 2 – Representação Básica de Arquitetura de 3 Camadas(MYERS,2004)

2.2.2 Estratégias de Teste

Além de compreender a operação dos diversos componentes de uma aplicação web, a elaboração de um documento de requisitos especificando a funcionalidade e o desempenho desejado é critico para o sucesso da realização dos testes. O teste necessita de uma estratégia para “dividir e conquistar”, e a divisão

em três camadas já permite identificar áreas-alvo para os testes. Cada camada possui características próprias que podem auxiliar a construção de casos de teste (MYERS, 2004).

- Teste da Camada de Apresentação: testar a camada de apresentação consiste em encontrar erros na interface gráfica que comprometam a qualidade e robustez da aplicação. Os principais pontos a serem considerados na camada 1 são:

- *Teste de conteúdo:* estética, fontes, cores, erros ortográficos e informações incorretas são os alvos.
- *Arquitetura do site:* erros navegacionais, links quebrados e páginas ausentes são procuradas.
- *Ambiente do usuário:* verificação da compatibilidade do navegador e do sistema operacional.

- Teste da Camada de Negócios: encontrar erros na lógica de negócios da aplicação. É muito similar ao teste de aplicações convencionais, inclusive quanto ao uso de *drivers* e *stubs*. A seguir estão algumas características que devem ser consideradas:

- *Desempenho:* teste para analisar se a aplicação contém especificações de desempenho documentadas (geralmente especificado em tempo de resposta ou taxas de transferência). O baixo desempenho dessa camada pode interferir na camada de apresentação.
- *Validação de dados:* assegurar que os dados coletados dos usuários são válidos.
- *Transações:* descobrir erros no processamento de operações, tais como verificação de e-mail e cálculos em geral.

- Teste da Camada de Dados: consiste em testar o quão seguro é o sistema de banco de dados utilizado para armazenar e recuperar informações. Os principais tipos de erros procurados por essa fase são:

- *Tempo de resposta:* identificar as operações que não satisfazem os requisitos de performance exigidos.
- *Integridade dos dados:* verificar se os dados foram armazenados corretamente e com precisão.
- *Tolerância à falhas e capacidade de recuperação:* sistemas mais complexos podem envolver a transferência de operações para outras bases de dados em caso de falha na principal. Neste caso, os testes

verificam se o tempo de transferência é o especificado e o que acontece com as transações pendentes neste momento. Também deve ser testada a capacidade de recuperar a base a partir de um backup.

2.2.3 Técnicas de Teste de Aplicações Web

Da mesma forma que um software tradicional, os processos de verificação e validação têm o propósito de checar respectivamente a qualidade da implementação e revelar não concordâncias com os requisitos do usuário. Técnicas estáticas diferem das dinâmicas quando não requerem execução. Elas trabalham em artefatos como design de documentos e código fonte HTML.

A seguir estão relacionadas algumas características das técnicas utilizadas no teste de aplicações Web.

- Verificação Estática: analisadores estáticos podem ser empregados para escanear as páginas HTML em um Web site e detectar possíveis erros e anomalias.

Controles estáticos podem ser focados em caminhos de navegação fornecidos ao usuário ou através dos fluxos de dados das informações obtidas pelo mesmo (RICCA & TONELLA, 2000 apud RICCA & TONELLA, 2001).

O modelo de um site pode ser analisado para determinar a presença de páginas inacessíveis, ou seja, páginas que estão disponíveis no servidor local, mas não podem ser acessadas por qualquer caminho começando da página inicial. Páginas fantasma são associações com links pendentes, que referenciam uma página não existente.

Uma informação útil sobre um Web site é o número mínimo de páginas que deve ser visitada antes de atingir o documento alvo. Informações sobre o menor caminho para cada página do site é um indicador de possíveis problemas para o usuário pesquisar um determinado documento, quando esse caminho é longo.

Re-calcular as análises estáticas acima descritas ao longo do tempo permite controlar a evolução da qualidade da aplicação (RICCA & TONELLA, 2001).

- Validação Dinâmica: aqui se encaixa o teste caixa-branca: a estrutura interna de uma aplicação Web é acessada para medir a cobertura que um determinado conjunto de testes atinge, em relação a um determinado critério de teste.

Um caso de teste para uma aplicação Web é uma seqüência de páginas a serem visitadas mais os valores de entrada que devem ser fornecidos para páginas que contenham formulários. Por isso, pode ser representada como uma seqüência de URLs especificando as páginas de perguntas e, se necessário, os valores a atribuir à entrada de variáveis. Execução consiste em solicitar o servidor Web para as URLs na seqüência e armazenar as páginas de saída. Diferentemente de softwares tradicionais, a seleção dependente pode ser forçada por escolher o *hyperlink* associado, sem ter de voltar para monitorar as condições de entrada de valores (exceto para arestas condicionais).

A seguir são apresentados alguns tipos de testes caixa-branca.

-Teste de Hyperlink: o Teste de *Hyperlink*, ou Teste de Link, testa se cada link de cada página do site é acessado pelo menos uma vez para que seja testado se este está devidamente conectado para que o conteúdo seja corretamente direcionado, e que não aconteça um re-direcionamento fantasma, ou seja, um link quebrado ou até mesmo um link que direcione a uma página que não traga o caminho esperado. Em aplicações Web, o Teste de Link tem um valor significativo, pois existe nas aplicações uma quantidade enorme de caminhos a serem seguidos, que quando implementados de forma errônea, prejudicam diretamente a qualidade da aplicação.

- Teste de Validação de HTML: o validador HTML informa o número da linha onde ocorreu erro no código e isto auxilia na localização do problema. O validador confere o código linha por linha começando da primeira linha. Isto implica que quanto mais no início do código um erro for detectado mais linhas subseqüentes ele afetará. Uma boa técnica para correção de erros consiste em corrigir o primeiro erro que apareceu e revalidar o código. É freqüente a correção de um erro trazer como conseqüência a correção de mais outros, apontados logo a seguir no código (DUBOST, 2002).

-Teste de Definição-uso: consiste na navegação de todos os caminhos de cada definição de uma variável para cada utilização da mesma, formando uma dependência de dados (BEIZER, 1990).

-Teste de Todos os Usos: navegação de pelo menos um caminho de cada definição de uma variável para cada utilização da mesma (BEIZER, 1990).

-Teste de Todos os Caminhos: cada caminho no site é acessado em alguns casos de testes, pelo menos uma vez (BEIZER, 1990).

Os critérios definição-uso e todos os caminhos são muitas vezes impraticáveis, uma vez que existem normalmente infinitos caminhos em um site, se loops estiverem presentes. Elas podem ser satisfeitas se forem impostas restrições adicionais sobre os caminhos a serem considerados (BEIZER, 1990).

- Teste de Regressão: o teste de regressão é um teste seletivo, de um software que foi modificado ou de iterações anteriores. Tem como propósito garantir que qualquer falha tenha sido reparada e que nenhuma operação que funcionava anteriormente tenha falhado após os reparos, ou seja, que as novas características adicionadas não criaram problemas com as versões anteriores ou com outros sistemas. Seu objetivo é verificar se as alterações realizadas geraram alguma inconsistência no aplicativo ou em outros sistemas. Utiliza como critério de avaliação os *scripts* desenvolvidos anteriormente, que são executados progressivamente para verificar que nenhuma falha foi introduzida após as mudanças (DATASUS, 2008).

Ao executar os casos de teste para uma aplicação Web, nem todas as páginas são igualmente de interesse. Páginas estáticas que não contenham formulários podem ser ignoradas, pois não recolhem informações fornecidas pelo usuário, processam ou apresentam resultados. Elas contêm informações fixas que não necessitam ser examinadas durante a validação dinâmica.

2.3. Ferramentas de Teste de Aplicações Web

Para auxiliar a atividade de teste, deve-se utilizar uma ferramenta de apoio ao teste. Uma ferramenta de teste reduz a intervenção humana nos resultados obtidos. Essa ferramenta deve produzir casos de teste que mostrem o comportamento errado do programa e revelar os defeitos.

O desenvolvimento de ferramentas para o suporte à atividade de teste propriamente dita também é de fundamental importância, uma vez que essa atividade é muito propensa a erros, além de improdutiva, se aplicada manualmente, cita Jorge *et al.* (2002).

Jorge *et al.*(2002) cita ainda que a disponibilidade de ferramentas de teste oferece recursos para o desenvolvimento de estudos empíricos que visem a avaliar e a comparar os diversos critérios de teste existentes. Ferramentas de teste permitem a transferência de tecnologia para as indústrias e contribui para uma

contínua evolução de tais ambientes, fatores indispensáveis para a produção de software de alta qualidade.

Além disso, tais ferramentas auxiliam pesquisadores e alunos de Engenharia de Software a adquirir os conceitos básicos e experiência na comparação, seleção e estabelecimento de estratégias de teste (VINCENZI, 1998).

Segundo Pressman (2002), existem as seguintes categorias de ferramentas de teste de software:

- Aquisição de Dados: ferramentas que adquirem dados a serem usados durante o teste;

- Medição Estática: ferramentas que analisam o código-fonte sem executar casos de teste. Existem três tipos diferentes de ferramentas de teste estático, sendo ferramentas de teste baseado em código, linguagens especializadas de teste e ferramentas de teste baseado em requisitos;

- Medição Dinâmica: ferramentas que analisam o código-fonte durante a execução do programa, ou seja, interagem com o programa em execução. As ferramentas dinâmicas podem ser intrusivas, que modificam o software a ser testado, inserindo instruções extras, ou não-intrusivas, que usam um processador de hardware separado que roda em paralelo com o processador contendo o programa que está sendo testado;

- Simulação: ferramentas que simulam funções do hardware e outros dispositivos externos;

- Gestão de Testes: ferramentas que assistem o planejamento, desenvolvimento e controle de testes, geram e coordenam testes de regressão, realizam comparações que avaliam as diferenças entre as saídas esperada e real;

- Ferramentas de Cruzamento Funcional: ferramentas que cruzam as fronteiras das categorias precedentes.

Deve-se notar que muitas ferramentas de teste têm características que abrangem duas ou mais das categorias citadas.

2.3.1 Ferramentas para Teste de Aplicações Web

Como visto na seção anterior, as ferramentas de teste são classificadas em diferentes categorias, o que proporciona uma variedade de opções para o testador. Portanto, este tem a responsabilidade de definir corretamente qual o tipo de teste

suprirá corretamente suas necessidades para poder definir qual ferramenta será utilizada.

A seguir são apresentadas algumas ferramentas para Teste de Aplicação Web. Devido ao número elevado de ferramentas, neste trabalho serão abordadas somente algumas, dando ênfase às classificadas como software livre.

- actiWATE: em termos de desenvolvimento de testes, o actiWATE parece mais com uma biblioteca programável que é utilizada para desenvolvimento de testes automatizados. O actiWate não oferece uma linguagem de script especial, seu teste automatizado é feito em código Java, utilizando apenas a API do actiWATE. Essa ferramenta não necessita de um navegador de internet para executar seus testes, pois a esta já contém um emulador que age da mesma maneira que um navegador de internet convencional. A emulação do navegador traz a vantagem de utilizar a ação baseada na API provida pelo actiWATE. Isso significa que, ao invés de operar com solicitações HTTP e formulários ou páginas HTML, o teste automatizado executa um conjunto de ações (comportando-se como um usuário) com o navegador, com as janelas, os links, botões e campos de texto.

O teste baseado em ação é mais fácil de escrever e compreender, com isso ele pode contribuir para o teste de regressão, no entanto, o actiWATE enquadra-se na categoria de Teste Funcional.

Em caso de falha no teste, o actiWATE gera um relatório em formato HTML , que simplifica a localização do problema. Outra informação relativa ao processo de execução de teste é o armazenamento em arquivos de registro (Log) (ACTIWATE, 2007).

- Selenium: é uma ferramenta de para testes de aplicações Web, distribuído sob a “Apache License, Version 2.0”. Essa ferramenta foi desenvolvida por Jason Huggins, e é uma ferramenta Open Source, desenvolvida com base em uma aplicação que utilizava Python e Plone, simples, prática e intuitiva que atua como ferramenta para Teste Funcional.

A ferramenta pode ser usada em três modos (ANTUNES, 2008):

- *Selenium Core - (Modo direto):* Os testes são efetuados diretamente através do browser. As páginas de teste devem estar hospedadas no mesmo servidor que a aplicação a ser testada. Esta restrição/característica é função da segurança relativa à mesma origem requerida pelo javascript.

*-Selenium IDE - (Modo indireto - Plugin no browser)**:*** Os testes são efetuados através de um *plugin* instalado no navegador FireFox. Este *plugin* é um ambiente integrado de desenvolvimento. Permite gravar a navegação do usuário, e depois repeti-la à titulo de teste. Também permite exportar os testes gravados em outros formatos.

-Selenium RC - (Modo indireto - Programa de teste + Proxy): Os testes são efetuados através de um programa, que comanda o browser através de um proxy. Este programa pode ser escrito nas linguagens Java, C#, Perl, PHP, Python e Ruby.

- WebLoad: foi projetada especificamente para testes de carga e desempenho de aplicações Web. Tem padrões abertos como base, e utiliza JavaScript como script de linguagem, alem de fornecer suporte nativo para Web 2.0 e Ajax[1]. O WebLoad proporciona um ambiente robusto e compreensível para o teste de carga. Isso inclui um ambiente próprio de gravação, edição e depuração de scripts de teste, um ambiente de execução altamente eficiente para definição de parâmetros de carga (usuário virtual), execução e acompanhamento dos testes. Possui ainda uma ferramenta de relatórios para analisar e apresentar os resultados.

O WebLoad trabalha com versões open source e temporárias, sendo as temporárias, as versões que ainda estão em desenvolvimento, e quando prontas, ficam dispostas na comunidade WebLoad.org para serem atualizadas e carregadas como open source (WEBLOAD, 2008).

- Apache JMeter: o JMeter é uma aplicação *desktop* completamente desenvolvida em Java, projetada para analisar o comportamento funcional e medir o desempenho do Teste de Carga. Foi projetada inicialmente para aplicações Web, mas atualmente expandiu-se para outras funções de teste.

O JMeter pode ser usado para testar o desempenho tanto de recursos estáticos, quanto dinâmicos (arquivos, Servlets, bases de dados e consultas, Servidores FTP). Ele pode ser usado para simular uma carga pesada em um servidor de rede ou em um objeto, para testar sua força ou analisar o desempenho no âmbito de diferentes tipos de carga. Pode ainda ser usado para fazer uma análise gráfica de performance, ou para testar o comportamento do servidor, script ou objeto sob uma carga pesada concorrente (APACHE, 2007) .

[1] AJAX (acrônimo em língua inglesa de Asynchronous Javascript And XML) é o uso sistemático de tecnologias providas por navegadores, como Javascript e XML, para tornar páginas mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações.

- Xenu's Link Sleuth: essa ferramenta verifica links quebrados em Web sites. A verificação é executada em links normais, imagens, frames, plug-ins, planos de fundo, mapas de imagens locais, folha de estilo, scripts e applets Java. O Link Sleuth mostra uma lista continuamente atualizada de URL's que podem ser classificados em diferentes critérios. A ferramenta também emite relatórios de acompanhamento (HAUSHERR, 1997).

Algumas características adicionais:

- Reverificação de links quebrados (útil para erros temporários de rede);
- Formato de relatório simples, que permite a emissão para e-mails;
- Arquivo executável menor que 1mb;
- Cria um mapa do site.

-LinkScan: foi projetado para funcionar tanto em servidores de Internet quanto Intranet, e sua capacidade de teste pode atingir até 40.000 links por hora, pelo fato de utilizar-se de processos simultâneos multi-topicos. O LinkScan pode lidar com 250.000 páginas de uma só vez. É uma ferramenta paga, mas proporciona uma opção de teste gratuito que permite a verificação de até 10 páginas por hora e até 50 por dia. Há um limite de 200 Links por Documento. Essa opção é chamada QuickCheck, e é concebida como uma forma rápida para verificar a qualidade e identificar problemas em uma página da Web (ELSOP).

Tabela 1 – Comparativo das Ferramentas de Teste de Aplicação Web

FERRAMENTA	CATEGORIA	LINGUAGEM	DISTRIBUIÇÃO
ActiWATE	Teste Funcional	Java	Livre
Selenium	Teste Funcional	Java,C#,PHP,Phyton,Ruby	Livre
WebLoad	Teste de Carga e Performance	Javascript, Ajax	Partes Open Source, e partes Licenciadas
Apache JMeter	Teste de Carga e Funcional	Java	Livre
Xenu's Link Sleuth	Teste de Link	--	Livre
LinkScan	Teste de Link	--	Licenciada e Trial para teste

3 FERRAMENTAS DE TESTE EM APLICAÇÕES WEB

3.1 Considerações Iniciais

Em busca dos objetivos desse trabalho, além da revisão de literatura, foram realizadas algumas outras atividades, tais como:

- Seleção de uma ferramenta de teste e de uma aplicação web básica para realização de teste funcional;
- Seleção de outra ferramenta e de uma página web para aplicação de teste de performance;
- Análise de resultados: avaliação da eficiência do uso de ferramentas, da eficácia dos testes e ganho de produtividade obtidas a partir de testes automatizados.

As próximas seções abordam estes tópicos.

3.2 Teste Funcional com a Ferramenta Selenium IDE

A ferramenta Selenium IDE é um ambiente integrado de desenvolvimento de testes automatizados que permite ao usuário a gravação, edição e depuração de testes.

O Selenium IDE é um *plugin* do navegador Firefox que permite a gravação da navegação realizada pelo usuário para que depois esse script gravado possa ser executado a título de teste. A ferramenta utiliza-se desse script gerado e emula um usuário virtual, renderizando a aplicação através de sua IDE, em busca de erros de funcionalidade. Durante a gravação do script, a ferramenta possibilita ainda a inserção manual de comandos que não são capturados durante a navegação do usuário, para testar componentes que não são afetados diretamente com a navegação da página, como por exemplo a presença de textos e de botões.

A maior característica positiva de se usar a ferramenta Selenium IDE, é que esta dispõe ainda da capacidade de gravar esses scripts gerados durante a navegação tanto para exportá-lo para outras linguagens como C#, RUBY, PHP, quanto para reutilização do script gerado pra testar novamente a aplicação depois que ela tenha sofrido alguma modificação ou correção encontrada em uma análise anterior. Essa funcionalidade é conhecida como *Record & Playback*, que é a

gravação e reprodução do teste tantas vezes quanto for necessário, utilizando-se o mesmo caso de teste desenvolvido inicialmente. Isso faz com que a ferramenta gere produtividade, pois se a cada modificação de código, houver a necessidade de se desenvolver toda a gravação do teste novamente, o processo se tornaria ineficiente.

3.2.1 Aplicação da Ferramenta

Para análise da ferramenta foi utilizada uma aplicação web simples, que lê duas datas e calcula o número de dias transcorrido entre elas. A Figura 3 mostra a interface da aplicação.

Figura 3 – Interface da aplicação web a ser testada

Observando que o objetivo deste trabalho é a verificação da eficiência e eficácia da utilização de ferramentas de teste automatizada, e não da qualidade e da integridade da aplicação em questão, inicialmente a aplicação foi utilizada com um erro de código artificial gerado pelo orientador deste trabalho, que afetou o resultado esperado. O erro introduzido era desconhecido pelo testador, o autor deste trabalho. Posteriormente, o defeito foi removido e a aplicação foi novamente testada com o script gerado, de maneira a forçar a mudança no resultado encontrado.

A Tabela 2 apresenta os casos de teste que foram utilizados durante o processo.

Tabela 2 – Casos de teste da aplicação web DATA

DADOS DE TESTE	RESULTADO ESPERADO
10/10/2008, 15/10/2008	Dias decorridos: 5
Clicar no link CONTINUAR	Redirecionar à Página Inicial

O processo de teste consiste em navegar pela aplicação web com o Selenium IDE ativo capturando em um script todas as informações necessárias ao teste.

Figura 4 – Navegação da aplicação para capturar o script de teste

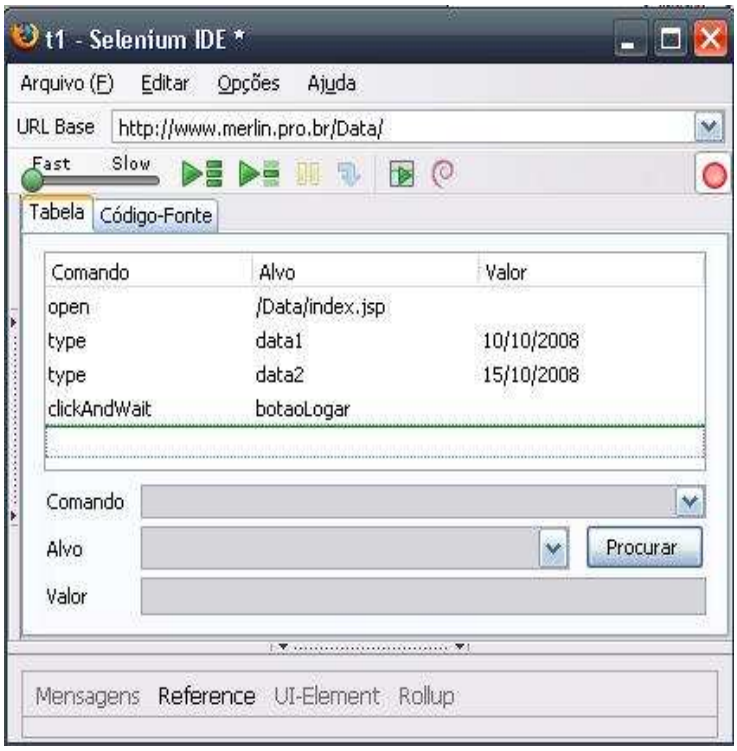

Figura 5 - Script capturado durante a navegação

Perante a gravação do script mostrado na Figura 5, nota-se uma certa escassez de comandos para tornar o teste consistente, sendo assim, a seguir, está disposto o script de teste com uma inserção de comando feita manualmente para melhor extração de resultados dessa aplicação.

Para execução do teste funcional é necessária a inserção manual de comandos. No caso, é preciso verificar o resultado do cálculo, o que pode ser feito pela análise do conteúdo retornado. O comando que tem essa função é identificado por *verifyTextPresent*, cujo funcionamento consiste em verificar a existência de um determinado texto na tela. Este comando permite informar à ferramenta o resultado esperado para o teste, que no caso é a diferença entre as datas.

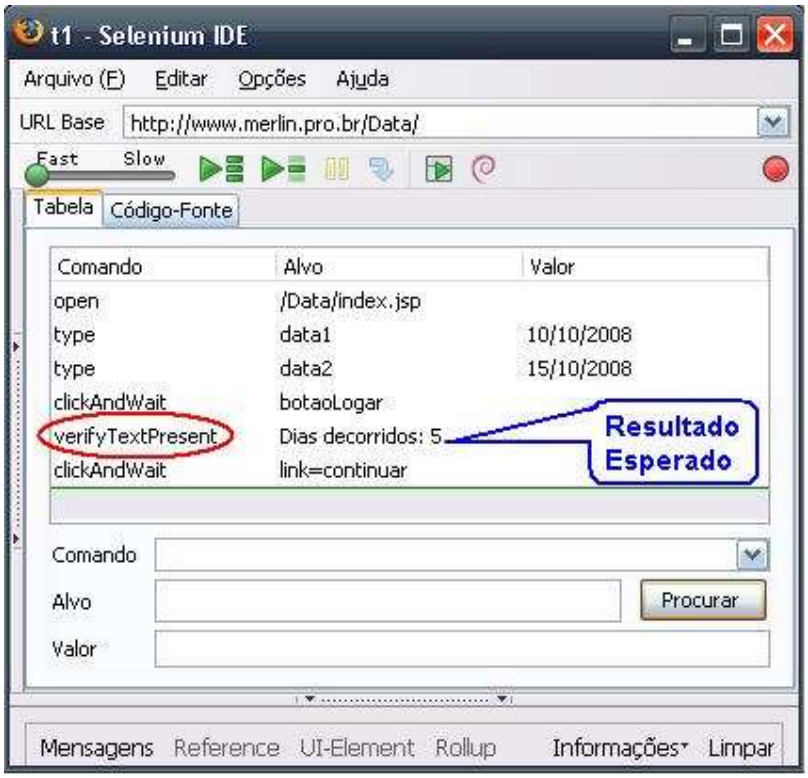

Figura 6 – Inserção de comando para teste do resultado

A Figura 7 mostra o resultado da execução automática do script de teste gerado em relação ao código com erro.

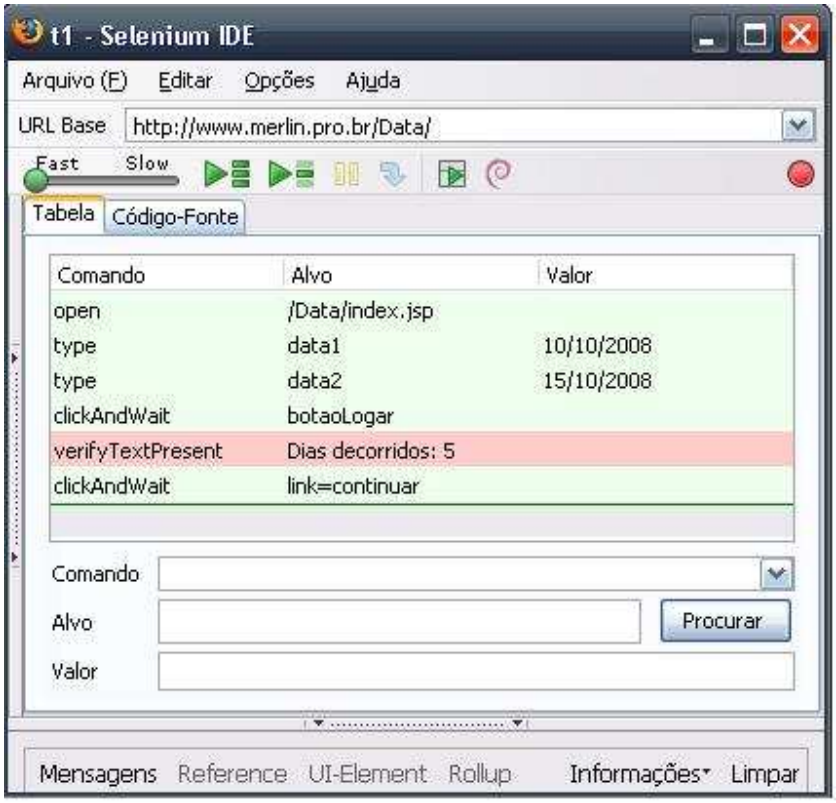

Figura 7 – Execução do teste automatizado pela ferramenta

Seguindo os casos de teste especificados, observa-se que foi detectado um erro no resultado esperado.

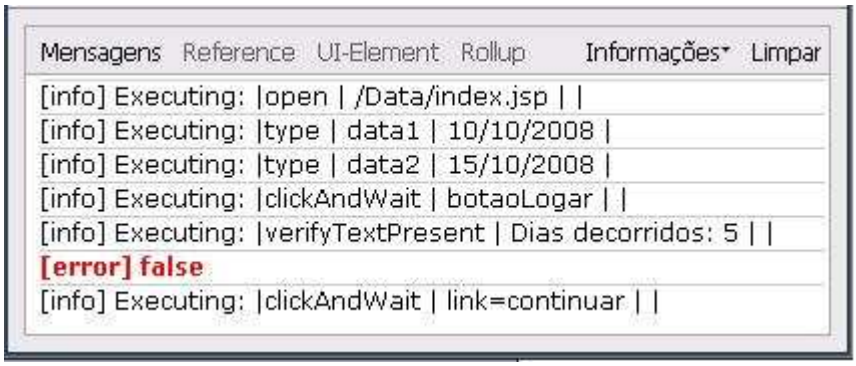

Figura 8 – Resultado do primeiro teste gerado pela ferramenta

De acordo com o caso de teste especificado, o resultado esperado era de **Dias decorridos: 5**, no entanto, a aplicação gerou um resultado de **Dias decorridos: 4,** como pode ser observado na Figura 4, ficando evidente, assim como na Figura 7, que a linha em vermelho aponta erro na aplicação.

Utilizando-se da principal função da ferramenta de testes automatizados, depois que a aplicação foi corrigida, o teste foi re-executado a partir do script já gravado anteriormente na intenção de encontrar o mesmo erro, e principalmente, com o intuito de verificar se a correção de um erro não acarretou na geração de

novas anomalias no código da aplicação. Esse tipo se verificação é conhecido como Teste de Regressão, apresentado na seção 2.2.3 deste trabalho, e caracteriza-se pelo fato de testar novamente a aplicação depois que ela tenha sido modificada.

A Figura 9 apresenta a segunda sessão de testes.

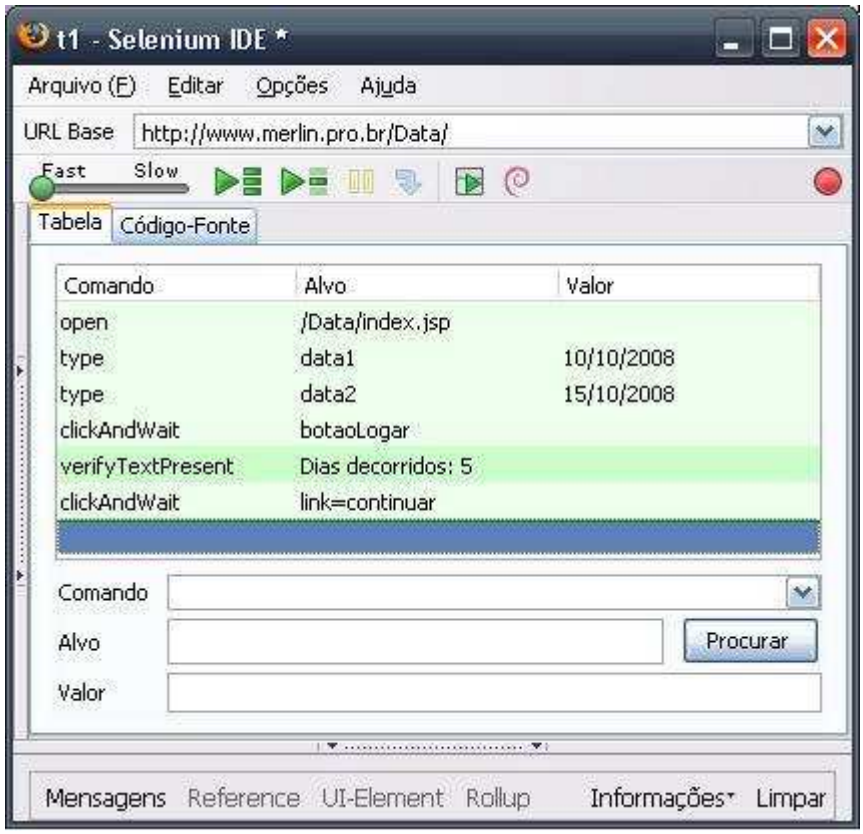

Figura 9 – Teste após a aplicação ser corrigida

Os resultados obtidos a partir da segunda sessão de testes com o mesmo script de teste da sessão anterior são apresentados na Figura 10.

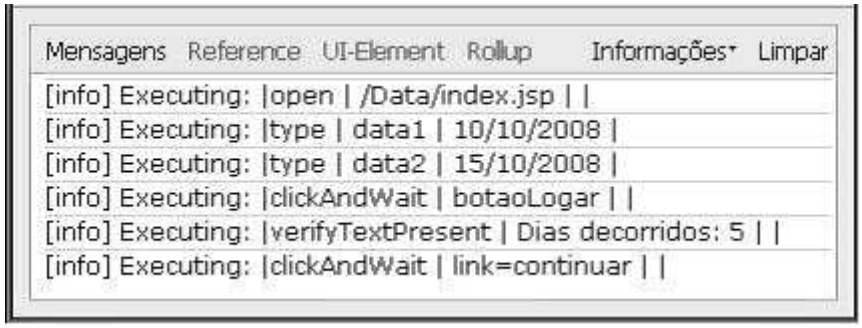

Figura 10 – Resultado do segundo teste gerado pela ferramenta

A partir disso pode-se notar, de acordo com a linha verde em destaque na Figura 9, que a correção da aplicação foi efetuada, e o teste foi realizado com

sucesso, não detectando a falha ocorrida na primeira sessão de testes, assim como nenhuma outra anomalia.

3.3 Teste de Desempenho com JMeter

A Seção 3.2 apresentou a Ferramenta Selenium, abordando os requisitos funcionais. No entanto, no processo de desenvolvimento de software, além dos requisitos funcionais, devem ser considerados os requisitos não funcionais.

Requisitos não-funcionais não dizem respeito às funções exercidas pelo software, mas determinam como essas funções devem ser oferecidas (DIAS & MENNA, 2008).

Um exemplo de requisito não funcional é o nível de desempenho; o usuário deve ter objetivos para o sistema em termos de uso de memória e tempo de resposta.

O objetivo do teste de desempenho é verificar se a aplicação satisfaz todos os requisitos de desempenho especificados através de possíveis gargalos ou insuficiências.

O teste de desempenho possui várias finalidades, tais como comparar dois sistemas com o intuito de identificar o mais eficiente, medir qual parte do sistema esta comprometendo o desempenho, e ainda verificar o comportamento do sistema quando muitos usuários o acessam simultaneamente.

No documento de requisitos, os objetivos de desempenho devem ser especificados claramente pelos usuários/clientes, e devem ser indicados no plano de teste da aplicação. Os objetivos devem ser quantificados de modo que a aplicação responda a uma consulta em uma quantidade de tempo aceitável, e que esse tempo desejado seja caracterizado de maneira quantitativa. Decorrente disso, ao final do teste por exemplo, os testadores terão conhecimento do tempo de resposta real em segundos(minutos, horas) e do número de transações processadas por um período de tempo.

3.3.1 Medidas de Desempenho

Segundo Dias & Menna (2008), a definição de desempenho é a maneira de como um sistema se comporta. Isto é, o desempenho de um sistema é determinado

por suas características de execução. Portanto, avaliar o desempenho de um sistema demanda definir quais características comportamentais devem ser consideradas. Para sistemas computacionais, em geral consideram-se quatro fatores para medida de desempenho:

- *Vazão (throughput):* taxa de atendimento de pedidos pelo sistema.

- *Utilização:* fatia de tempo em que o sistema permanece ocupado, atendendo a pedidos;

- *População:* quantidade de clientes a serem atendidos em um determinado instante;

- *Tempo de resposta:* intervalo de tempo entre o pedido e o início/conclusão do serviço (DIAS & MENNA, 2008).

3.3.2 A Ferramenta Apache JMeter

A ferramenta JMeter foi desenvolvida pelo grupo Apache, disponível em código aberto e projetada totalmente com tecnologia Java para utilização em teste de carga em serviços oferecidos por aplicações computacionais. O JMeter realiza também testes de desempenho e de caixa-preta. Uma de suas características é permitir execuções de planos de testes que podem ser configurados graficamente.

Na realização dos testes, a ferramenta dispõe de vários tipos de requisições e de *assertions* (validação dos resultados das requisições), alem de controladores lógicos de ciclo (*loop*) e controles condicionais utilizados na elaboração de planos de testes que correspondem a testes funcionais.

O JMeter disponibiliza também um controle de *threads*, chamado *Thread Group*, no qual é possível configurar o número de *threads* (usuários), a quantidade de vezes (*loop count*) que cada *thread* será executada e o intervalo de tempo(*ramp-up period*) entre cada execução. E por fim, existem diversos *listeners*, que se baseando nos resultados das requisições ou dos *assertions*, podem ser usados para gerar gráficos e tabelas. Os componentes no JMeter são recursos que podem ser utilizados para criar rotinas de testes para aplicações. Por exemplo:

-Test Plan: para realização de qualquer teste utilizando o JMeter, é necessário criar um Test Plan incluindo os elementos do teste. Estes elementos podem ser:

- *Thread Group:* este é ponto inicial, em que todos os outros elementos do *Test Plan* devem estar sob este. Como o próprio nome ressalta, este controla os *threads* (usuários) que serão executados durante o teste;

- *Controllers*: estes são divididos em dois grupos *Samplers* e *Logic Controllers*. *Samplers* são controladores pré-definidos para requisições específicas. Podendo ser customizada com a inserção de configurações (*Configurations*), *Assertions* e etc. *Logic Controllers* são controladores mais genéricos. Podendo ser customizada com a inserção de outros *controllers*, *configuration elements*, *assertions*, etc.

- *Listeners*: estes são os elementos que fornecem acesso as informações obtidas pelo JMeter durante os testes.

- *Timers*: por padrão, o JMeter faz requisições sem pausas entre elas. Os *timers* são utilizados para incluir pausas entre as requisições.

- *Assertions*: usado para verificar se a resposta obtida na requisição é a esperada.

- *Configuration Elements:* embora não faça requisições (exceto para HTTP Proxy Server), este elemento pode adicionar ou modificar as requisições.

3.3.3 A Aplicação Web Escolhida

Pela facilidade de acesso, a aplicação web escolhida para a realização do teste de desempenho com a ferramenta JMeter foi uma aplicação da própria universidade, para controle das atividades de pesquisa e extensão. Utilizou-se uma página de livre acesso, pois de outra forma o teste realizado poderia ser interpretado como tentativa de invasão por sistemas de detecção de intrusos.

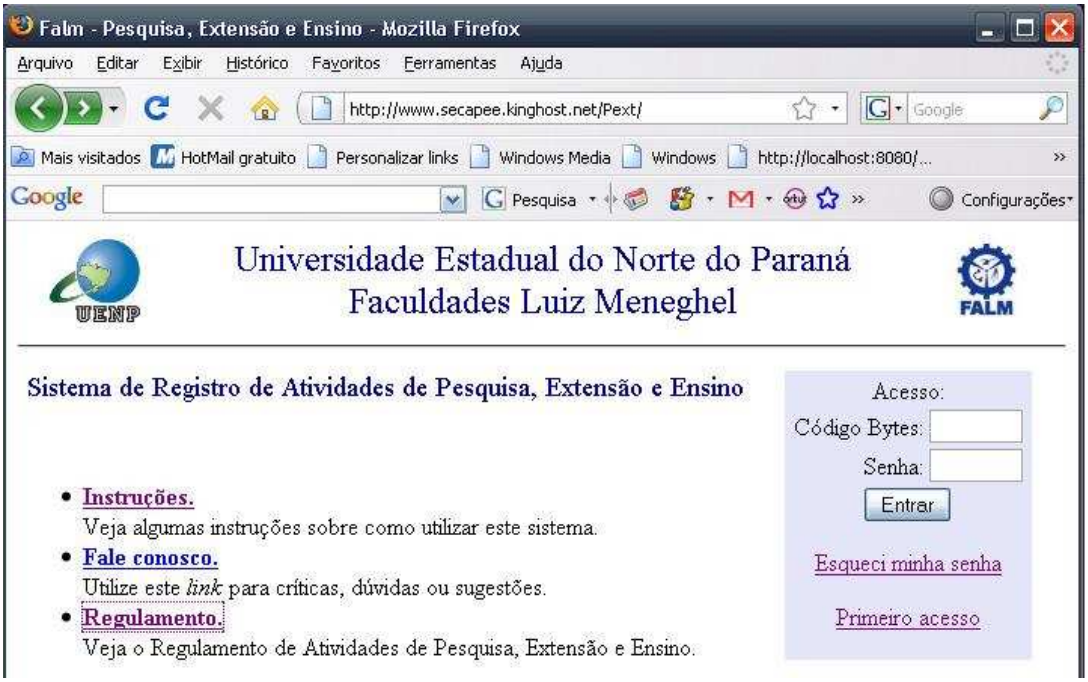

Figura 11 – Página inicial da aplicação web a ser testada

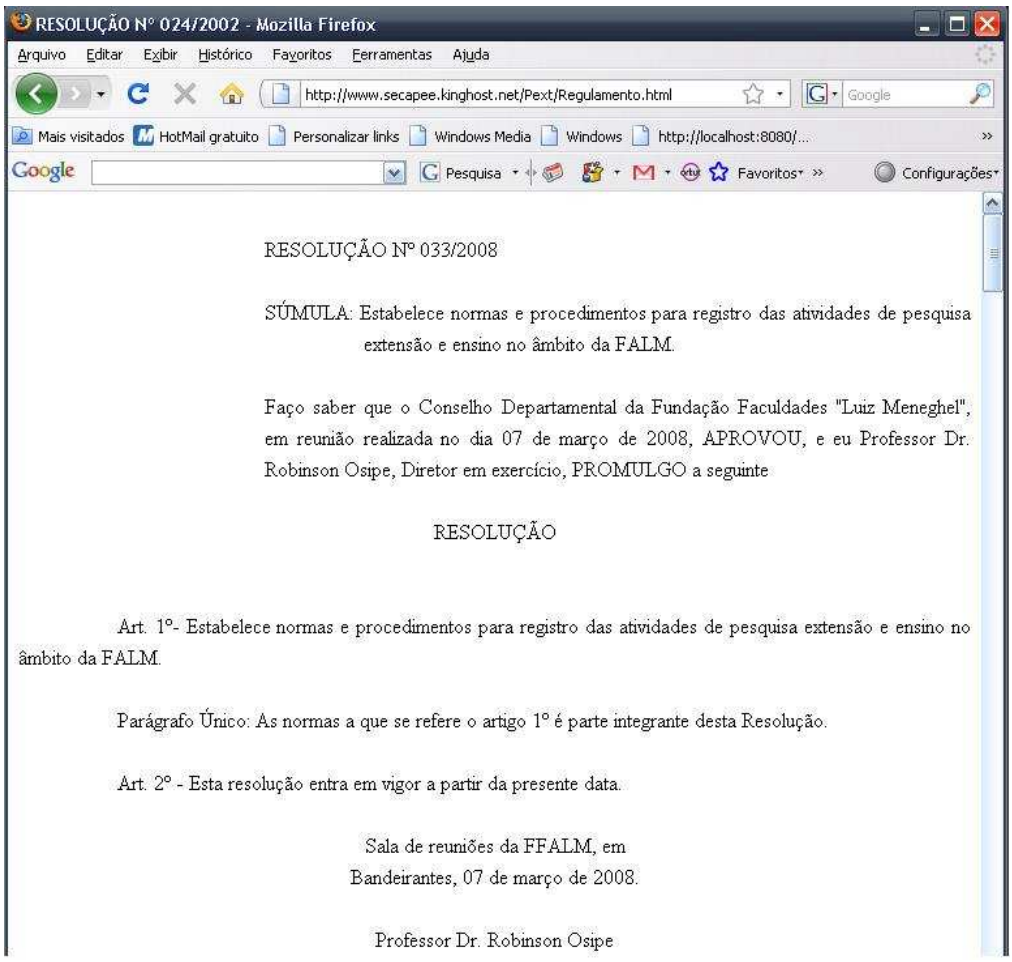

Figura 12 – Página acessada da aplicação web para realização do teste

3.3.4 A Realização do Teste

Para a realização do teste, foi feito o acesso à página inicial (Figura 11) e à partir dela, através do *link* “Regulamento” um documento de texto foi carregado (Figura 12). A função do teste de desempenho empregada aqui é medir a performance da aplicação em relação à quantidade variada de usuários acessando-a simultaneamente. A princípio a simulação foi efetuada com um usuário acessando, em seguida, com dez, e por fim, uma simulação com cem usuários. Vale lembrar que todas as simulações são idênticas em relação à requisição/resposta, utilizando o mesmo script, apenas modificando-se o número de usuários a cada teste.

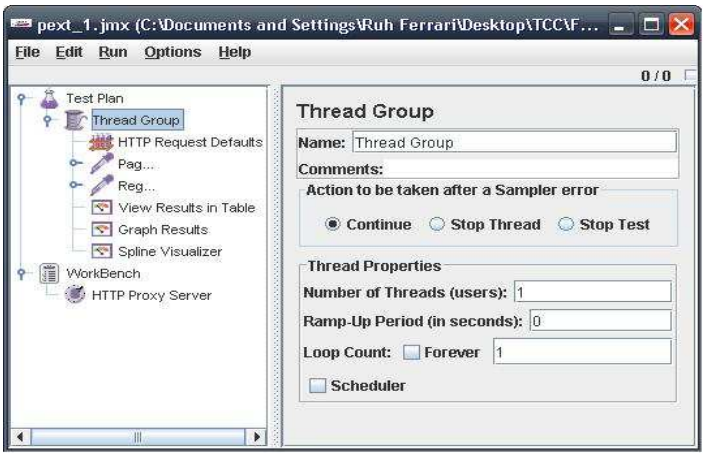

Figura 13 – Interface da ferramenta para configuração das quantidades de usuário, período de tempo e quantidade de vezes

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
Pagina Inicial	1	163	163	163	163	163	0,00%	6,1/sec	19,8
Regulamento	1	65	65	65	65	65	0,00%	15,4/sec	49,5
TOTAL	2	114	163	163	65	163	0,00%	8,7/sec	28,1

Figura 14 – Relatório Agregado de desempenho com 1 usuário

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
Pagina Inicial	10	196	200	230	167	230	0,00%	40,5/sec	130,4
Regulamento	10	87	87	88	86	88	0,00%	59,9/sec	192,8
TOTAL	20	142	167	222	86	230	0,00%	59,7/sec	192,2

Figura 15 – Relatório Agregado de desempenho com 10 usuários

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
Pagina Inicial	100	1104	1131	1744	302	3171	0,00%	31,5/sec	101,5
Regulamento	100	436	365	586	65	1727	0,00%	34,2/sec	110,0
TOTAL	200	770	520	1679	65	3171	0,00%	61,8/sec	198,9

Figura 16 – Relatório Agregado de desempenho com 100 usuários

O Gráfico 1 ilustra os resultados obtidos a partir das 3 execuções do teste.

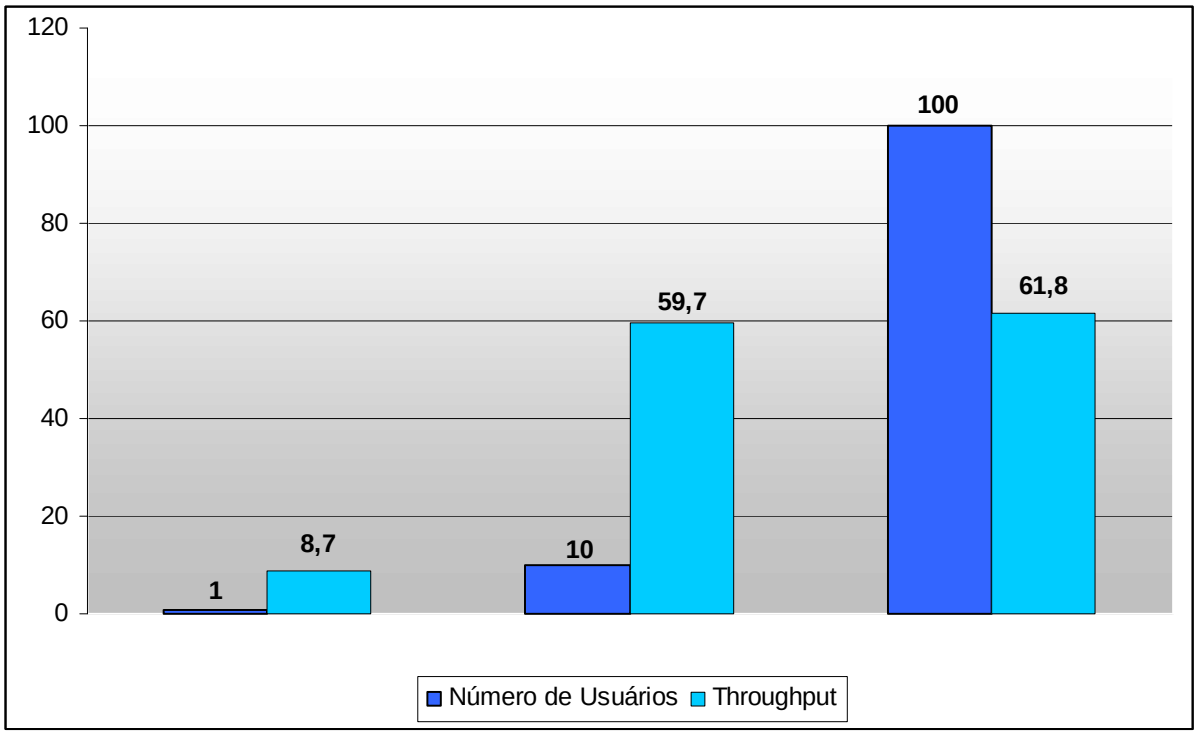

Gráfico 1 – Resultado da execução do teste com quantidades variadas de usuários

Conforme demonstrado no Gráfico 1, os resultados colhidos foram que, com 1 usuário acessando, a taxa de atendimento de pedidos pelo sistema (*throughput)* foi de 8,7 segundos, com 10 usuários a taxa foi de 59,7 segundos e por fim, com 100 usuários, a taxa obtida foi de 61,3 segundos.

4 RESULTADOS E DISCUSSÕES

Levando em consideração o teste funcional executado, a partir da ferramenta Selenium IDE, pôde-se obter resultados satisfatórios quanto ao uso da ferramenta, primeiramente pelo fato da eficácia na detecção de erros, mas principalmente pela agilidade com que o teste foi executado pela segunda vez, testando a aplicação novamente depois que ela sofreu alteração (correção, modificação). Essa principal satisfação deu-se pelo fato do tempo em que essa re-execução ocorreu, já que, no primeiro momento, o script de teste teve que ser capturado, sofrer alguns ajustes, e a partir daí, foi executado para a detecção de erros. No segundo momento, esse script de teste já estava pronto, somente tendo que ser iniciado para que o procedimento automático da ferramenta fosse disparado em busca de novos erros ou erros conseqüentes da modificação.

Esse ganho de produtividade com testes automatizados é essencial em dias atuais, em que o desenvolvimento de aplicações para web necessitam ser rápidos, flexíveis, e de qualidade.

É importante ressaltar aqui, que apesar de muito eficiente e produtiva, a ferramenta apresenta alguns pontos críticos a serem comentados, como por exemplo, a limitação do tipo de teste que ela executa, o teste funcional. Para o teste completo de uma aplicação são necessários outros tipos de teste, o que acarreta a necessidade de se utilizar outras ferramentas. Outra característica que vale ser citada é a dificuldade que foi encontrada para a compreensão inicial do funcionamento da ferramenta, já que documentações oficiais, manual de usuário ou tutoriais são, de certa forma, escassos até o presente momento.

O segundo tipo de teste efetuado, diz respeito ao teste de desempenho de uma aplicação web utilizando o JMeter. Seus resultados mostraram a diferença de tempo de vazão (*throughput*) que diferentes quantidades de usuários podem causar ao acessar a aplicação simultaneamente. O acesso de 1(um) usuário leva de 8,7 segundos na taxa de atendimento de pedidos pelo sistema (*throughput*). O acesso de 10 (dez) usuários tem taxa de 59,7 segundos, e a quantia de 100 (cem) usuários causa um efeito na taxa de 61,8 segundos. Sendo assim, pôde-se notar que a diferença entre 1 e 10 usuários é relativamente significante comparado com a diferença de acesso entre 1 e 100. Na primeira comparação há uma diferença de 9

usuários, com uma variação entre as taxas de 51 seg. A segunda comparação tem a diferença de 99 usuários em uma variação de tempo de 53,1.

A comparação é mostrada na tabela 3.

Tabela 3 – Comparação dos resultados obtidos com o teste

USUARIOS	*THROUGHPUT*	Diferença de 1 a 10		Diferença de 1 a 100	
1	8,7s				
10	51,7s	**Usuários**	***Throughput***	**Usuários**	***Throughput***
100	61,8s	**9**	**51s**	**99**	**53,1s**

Isso comprova que o servidor tem capacidade para suportar uma grande quantidade de acessos, mantendo o desempenho.

Os resultados obtidos pela ferramenta devem ser confrontados com a especificação de desempenho. Desta forma, pode-se verificar se o resultado encontrado é igual ao resultado esperado. Ocorre que, muitas vezes, os requisitos não-funcionais deixam de ser especificados e isto impede o teste destes requisitos. Em aplicações web, nas quais requisitos de segurança e desempenhos são notadamente importantes, estes requisitos deveriam ser especificados, assim como os requisitos funcionais.

O uso da ferramenta JMter então foi eficiente pelo fato de poder simular quantidades variadas de usuários acessando a mesma aplicação, em um único computador. Tal simulação seria impraticável sem o uso de uma ferramenta, pois haveria a necessidade de se montar uma rede e de se recrutar um grande número de pessoas para acessar a aplicação simultaneamente. Outro fator significativo a favor da ferramenta foi o uso do mesmo script de teste para poder simular as quantidades diferentes de usuários, garantindo, assim, boa produtividade na execução de testes automatizados.

As dificuldades encontradas aqui são semelhantes às encontradas para a ferramenta Selenium IDE comentada anteriormente. Porém esta última conta com um pouco mais de material de apoio disponível, provendo assim, uma maior facilidade para o aprendizado do seu uso. Importante ressaltar que além de teste de desempenho, a ferramenta JMeter disponibiliza outros tipos de teste, tais como teste de carga, de stress e de funcionalidade.

5 CONCLUSÃO

A necessidade de encontrar erros em uma aplicação web antes desta ser entregue ao usuário é de fundamental importância, e esta atividade pode ser realizada através do teste, que apesar de ser uma atividade que consome grande parte dos recursos alocados no desenvolvimento de uma aplicação, é indispensável para garantir a integridade e a qualidade da mesma. Com o intuito de minimizar a quantidade desses recursos utilizados pelo processo de teste no desenvolvimento de aplicações, diversas técnicas e critérios têm sido elaborados para apoiar essa atividade.

Visando avaliar a produtividade, a eficiência e a eficácia do uso de ferramentas de teste automatizadas, neste trabalho foi apresentado um estudo de caso para representar de maneira mais objetiva e prática a aplicação desses critérios. Inicialmente foram escolhidas duas ferramentas, uma para representar teste funcional, juntamente com uma aplicação web simples, e outra para realização de teste de desempenho, a partir de outra aplicação. Observando os resultados obtidos, é notável que a aplicação dessas ferramentas automatizadas foram satisfatórias (positivas), tanto na detecção de erros, provando sua eficácia, quanto no ganho de tempo ao utilizar a automação do teste, enfatizando assim sua eficiência. A grande vantagem da utilização de ferramentas automatizadas pode ser caracterizada pelo poder de testar grandes aplicações, de modo que o teste fique mais bem elaborado, evitando processos manuais, e podendo reutilizar-se do teste caso a aplicação seja modificada.

REFERÊNCIAS

ACTIWATE - Web application test environment, *How does actiWATE work?,* 2007. Disponível em: < http://www.actiwate.com/index.html >. Acesso em 06 abr 2008.

ANTUNES O. J., *Testando aplicações Web com Selenium.* 24 jan 2008. Disponível em: < http://log4dev.com/2008/01/24/testando-aplicacoes-web-com-selenium/>. Acesso em 27 fev 2008.

APACHE. The Apache Jakarta Project, *Apache JMeter,* 29 set 2007. Disponível em < http://jakarta.apache.org/JMeter/ >. Acesso em: 07 abr 2008.

BEIZER, B*. Software Testing Techniques*. Van Nostrand Reinhold Company, New York, 2nd edition, 1990.

CONALLEN, J.. *Building Web Applications with UML*. Addison-Wesley Publishing Company, Reading, MA, 2000.

DATASUS, *Processo de Testes de Software.* Disponível em: <http://pts.datasus.gov.br/ >. Acesso em: 05 abr 2008.

DIAS L. C. & MENNA R. S.; *Teste de Desempenho a Partir de Modelos Uml Para Componentes de Software*, PUC-RS, Porto Alegre, 2008.

DUBOST K., *My Web site is standard! And yours?.*W3C, 2002. Disponível em: <http://www.maujor.com/w3ctuto/webquality.html#Tools >. Acesso em: 05 abr 2008.

ELSOP - Electronic Software Publishing Corporation, *LinkScan.* Disponível em: < http://www.elsop.com/linkscan/ >. Acesso em: 07 abr 2008.

GIL, A. C. *Como Elaborar Projetos de Pesquisa*. 4ª ed. São Paulo: Editora Atlas, 2008.

HAUSHERR, T., *Xenu's Link Sleuth*, 1997. Disponível em: < http://home.snafu.de/tilman/xenulink.html#Description >. Acesso em: 07 abr, 2008.

HOWDEN, W. E*. Software Engineering and Technology: Functional Program Testing and Analysis*. McGrall-Hill Book Co, New York, 1987.

JORGE, R. F., VINCENZI, A. M. R., DELAMARO, M. E., MALDONADO, J. C.; *Relatório dos Operadores de Mutação implementados nas ferramentas Proteum e PROTEUM/IM*, ICMC - Usp, São Carlos, SP, 2002.

MALDONADO J. C.. Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software. PhD thesis, DCA/FEE/UNICAMP, Campinas, SP, July 1991.

MYERS, G.J., *The Art of Software Testing.* John Wiley & Sons, INC. 2nd ed. Hoboken, New Jersey, 2004.

PRESSMAN R. S.. *Engenharia de Software.* McGraw-Hill, 5 ed, Rio de Janeiro, 2002.

RAPPS, S. & WEYUKER, E. J.. *Selecting software test data using data flow information.*IEEE Transactions on Software Engineering, SE-11(4):367–375, April 1985.

RICCA F. & TONELLA, P. *Analysis and Testing of Web Applications.* 2001. ITC-irst Centro per la Ricerca Scientifica e Tecnologica, I-38050 Povo (Trento), Italy.

SOMMERVILLE I..*Engenharia de Software.*Addison Wesley,2003.

VINCENZI A. M. R., MALDONADO J. C., DELAMARO M. E., SPOTO E. S. E WONG W. E..*Component Based Software Quality: Methods and Techniques,* volume 2693*of Lecture Notes in Computes Science,* chapter Component-Based Software: An Overview of Testing, pages 99-127. Springer Verlag, New York, NY, June 2003. (A. Cechich and M. Piattini and A. Vallecillo ed.).

VINCENZI A. M. R.; *Subsídios para o Estabelecimento de Estratégias de Teste Baseadas na Técnica de Mutação*, Dissertação de Mestrado, ICMC/USP, Novembro, 1998.

WEBLOAD, *WebLoad overview,* 2008. Disponível em: <http://www.webload.org/overview.html >. Acesso em 06 abr 2008.