

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO
DE SISTEMAS

RAFAEL BARBOSA FERREIRA

ESTUDO EXPERIMENTAL COM TESTES PARA ENGENHARIA *WEB*

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA

2013

RAFAEL BARBOSA FERREIRA

ESTUDO EXPERIMENTAL COM TESTES PARA ENGENHARIA *WEB*

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – CSTADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Msc. Alessandra B. G. Hoffmann

MEDIANEIRA

2013



TERMO DE APROVAÇÃO

ESTUDO EXPERIMENTAL COM TESTES PARA ENGENHARIA *WEB*

Por

Rafael Barbosa Ferreira

Este Trabalho de Diplomação (TD) foi apresentado às 14:40 h do dia 23 de agosto de 2013 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. Os acadêmicos foram argüidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof. Alessandra B. G. Hoffman
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Me. Fernando Schütz
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Msc. Alan Gavioli
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Juliano Lamb
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTO

Agradeço primeiramente a minha família por todo apoio e incentivo prestado e sempre estar presente em todo meu processo de formação acadêmica.

Agradeço a todos os professores da UTFPR – Câmpus Medianeira, que em momento algum mediram esforços para auxiliar-me durante todo o curso de formação, ensinando e esclarecendo dúvidas.

Agradeço também a minha professora orientadora Alessandra B. G. Hoffmann que esteve sempre a disposição para o acompanhamento deste trabalho, sugerindo idéias, correções e fornecendo materiais para estudo.

RESUMO

FERREIRA, Rafael Barbosa. ESTUDO EXPERIMENTAL COM TESTES PARA ENGENHARIA *WEB*. Trabalho de Diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira, 2013.

Este trabalho apresenta conceitos sobre a Engenharia *Web* com foco em testes, que é uma parte do desenvolvimento de uma aplicação *Web* importante para que seja produzido uma aplicação confiável, segura para o cliente. Para a realização dos testes de uma *WebApp* foi utilizado o site do Meditec *In Technology* (www.meditec.net.br) juntamente com as ferramentas para teste de unidade (Selenium IDE e TestComplete) e de carga (Apache Jmeter e *Neoload*). A interface foi testada com a elaboração de um questionário onde os usuários puderam avaliar e fazer considerações.

Palavra-chave: Teste, *WebApp*, Ferramentas

RESUMO EM LINGUA ESTRANGEIRA

FERREIRA, RAFAELBARBOSA. EXPERIMENTAL STUDY WITH TESTS FORENGINEERING *WEB*. Trabalho de Diplomação (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira, 2013.

This work presents concepts on Web Engineering focusing on testing, which is a part of developing a web application that is important for an application produced reliable, safe for customer. For the testing of a *WebApp* site was used Meditec In Technology (www.meditec.net.br) along with the tools for unit testing (Selenium IDE and TestComplete) and load (Apache Jmeter and *Neoload*). The interface has been tested with the development of a questionnaire where users could evaluate and make considerations.

Keyword: Test, *WebApp*, Tools

LISTA DE SIGLAS

HTML	<i>HyperText Markup Language</i>
NIST	<i>National Institute Of Standards And Technology</i>
SQA	<i>Software Quality Assurance</i>
URL	<i>Uniform Resource Locator</i>
WebApp	<i>Web application</i>

LISTA DE FIGURAS

Figura 1: Processo de Engenharia <i>Web</i>	17
Figura2 :Projeto Padrão.....	21
Figura 3 Ciclo de Avaliação.....	22
Figura 4: Atributos de qualidade.....	23
Figura 5: Custos de um projeto.....	25
Figura 6: Pirâmides de uma construção.....	28
Figura 7: Camadas entre cliente e servidor.....	29
Figura 8: Conjunto de graus.....	33
Figura 9: Código de teste.....	40
Figura 10: Relatório de testes com eclipse e Junit.....	41
Figura 11: Trecho de código usando Selenium RC e Junit.....	42
Figura 12: Selenium IDE.....	43
Figura 13: Apache JMeter.....	44
Figura 14: TestComplete.....	45
Figura 15: Testando Sikuli.....	46
Figura 16: Teste com Badboy.....	47
Figura 17: Árvore de atributos.....	49
Figura 18: Página inicial da aplicação.....	51
Figura 19: Erro de ortografia.....	52
Figura 20: Alinhamento do texto.....	53
Figura 21: Recorte do site.....	53
Figura 22: Mensagens de alerta.....	54
Figura 23: Teste com Selenium IDE.....	55
Figura 24: TestComplete.....	56
Figura 25: Ações gravados no TestComplete.....	56
Figura 26: Gráfico de resposta 1.....	57
Figura 27: Gráfico de resposta 2.....	58
Figura 28: Gráfico de resposta 3.....	58
Figura 29: Gráfico de resposta 4.....	59
Figura 30: Gráfico de resposta 5.....	59
Figura 31: Gráfico de resposta 6.....	60

Figura 32: Gráfico de resposta 7.....	60
Figura 33: Gráfico de resposta 8.....	61
Figura 34: Gráfico de resposta 9.....	61
Figura 35: Gráfico de resposta 10.....	62
Figura 36: Gráfico de resposta 11.....	62
Figura 37: Compatibilidade de navegadores.....	63
Figura 38: Navegador Internet Explorer.....	64
Figura 39: Teste carga com Jmeter.....	65
Figura 40: Tabela de resultados 1.....	65
Figura 41: Gráfico de resultados 1.....	66
Figura 42: Tabela de resultados 2.....	66
Figura 43: Gráfico de resultados 2.....	67
Figura 44: Resumo teste 1.....	68
Figura 45: Gráfico teste 1.....	68
Figura 46: Resumo teste 2.....	69
Figura 47: Gráfico teste 2.....	69
Figura 48: Resumo teste 3.....	70
Figura 49: Gráfico teste 3.....	70

SUMÁRIO

1	INTRODUÇÃO.....	12
1.1	OBJETIVO GERAL.....	13
1.2	OBJETIVOS ESPECÍFICOS	13
1.3	JUSTIFICATIVA	13
2	FUNDAMENTAÇÃO TEÓRICA.....	15
2.1	ENGENHARIA DE SOFTWARE	15
2.2	ENGENHARIA <i>WEB</i>	16
2.2.1	Um projeto <i>WebApp</i>	18
2.2.2	Projeto de interface	19
2.2.3	Qualidade de software	22
2.2.4	Custo de qualidade.....	24
2.3	TESTES PARA APLICAÇÃO <i>WEB</i>	26
2.3.1	Visão Geral	27
2.3.2	Teste de conteúdo	28
2.3.3	Teste de interface de Usuário.....	30
2.3.4	Teste de Usabilidade.....	31
2.3.5	Teste de Compatibilidade	33
2.3.6	Teste de Navegação	34
2.3.7	Teste de Configuração	34
2.3.8	Teste de Segurança	36
2.3.9	Teste de Desempenho	37
2.3.10	Teste de carga	38
2.3.11	Teste de esforço	39
2.4	FERRAMENTAS DE TESTES PARA ENGENHARIA <i>WEB</i>	39
2.4.1	Testes automatizados com JUnit	40
2.4.2	Selenium RC.....	41
2.4.3	Selenium IDE.....	42
2.4.4	Apache Jmeter	44
2.4.5	TestComplete.....	44
2.4.6	SikuliApi.....	45

2.4.7	<i>Neoload</i>	46
3	MATERIAIS E MÉTODOS	48
3.1	MATERIAL.....	48
3.2	MÉTRICA	48
3.2.1	Métricas de Boehm, Brown e Lipow	48
3.3	MÉTODOS	50
4	RESULTADOS E DISCUSSÕES	51
4.1	TESTES	51
4.1.1	Teste de conteúdo	51
4.1.2	Teste de interface de Usuário.....	53
4.1.3	Teste de usabilidade.....	57
4.1.4	Teste de compatibilidade	62
4.1.5	Teste de carga e <i>stress</i>	64
5	CONCLUSÕES	72
5.1	CONSIDERAÇÕES FINAIS	72
5.2	TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO	73
6	REFERÊNCIAS BIBLIOGRÁFICAS	74

1 INTRODUÇÃO

A expectativa é de crescimento da Internet, automação e controle nas indústrias e empresas. Os sistemas estão cada vez mais voltados para *Web* e a qualidade exigida pelos clientes segue no mesmo ritmo. Uma das formas mais utilizadas e recomendadas para se garantir a qualidade de um software se dá a partir da realização de testes, que são usados para revelar a presença de defeitos (MYERS, 2004).

O processo de teste desempenha um papel crucial no processo de desenvolvimento global, no entanto está parte do desenvolvimento é negligenciada, muitos desenvolvedores testam o sistema somente depois da finalização de todo o processo de evolução da aplicação. O teste deve acompanhar diversas fases do ciclo de vida do desenvolvimento *Web*, com isso é possível garantir níveis ótimos de desempenho e seu bom funcionamento e evitar dispendiosas correções retroativas e possíveis riscos. Testar e validar um grande sistema *Web* é uma tarefa difícil e caro. Deve-se verificar desde a concepção até a implantação, manutenção e contínuo aprimoramento (MURUGESAN e GINIGE, 2005)

O teste de software é a investigação do aplicativo a fim de extrair informações sobre sua qualidade em relação à função que ele deve operar e conseguir detectar os possíveis defeitos. Um conceito interessante é utilizar pessoas e organizações diferentes para implementação e para a verificação, visto que a entidade de implementação tem visão de construção, não se preocupando muito com os erros, já a entidade de teste possui uma visão destrutiva, a procura de erros (MYERS,2004)

Os sistemas baseados em *Web* residem e interagem com diferentes sistemas, navegadores, plataformas de hardware e protocolos de comunicação. A busca de erros representam um desafio significativo para engenheiros *Web* (PRESSMAN e LOWE, 2009).

O presente trabalho aborda um estudo experimental através de uma fundamentação teórica acerca de Engenharia *Web*, Critérios, Métricas de qualidade e Testes em *WebApps* citando e utilizando programas que executem testes automatizados que identifiquem possíveis erros e validações. Os testes são realizados com o propósito de encontrar a qualidade dessa aplicação apresentando os dados obtidos, sendo possível discutir os resultados e apresentar as conclusões.

1.1 OBJETIVO GERAL

Desenvolver um estudo experimental sobre testes de softwares em aplicações *Web*.

1.2 OBJETIVOS ESPECÍFICOS

- Elaborar um referencial sobre Engenharia *Web*;
- Desenvolver um referencial sobre testes de *software* para *Web* e ferramentas automatizadas;
- Definir métricas e aplicação *Web* para os testes de software;
- Realizar testes de software em uma aplicação *Web* experimental;
- Apresentar resultados obtidos nos testes;

1.3 JUSTIFICATIVA

Empresas estão dependendo cada vez mais de sistemas que usam a Internet, sem falar em sistemas que dependem unicamente desses sistemas como o *e-commerce*. Conforme esse crescimento, conseqüentemente os números de sistemas que apresentam erros também aumentam. Entre esses defeitos podemos citar erros de navegabilidade, acessibilidade, erros de confiabilidade, problemas de segurança entre outros.

Uma empresa moderna exige adaptação, estratégias e regras que podem mudar rapidamente, assim a resposta deve ser rápida. Com isso, uma equipe de *Web* tem que seguir no mesmo ritmo, precisa enfatizar a agilidade. A Engenharia *Web* aborda todas as práticas e fluxo de processo da Engenharia de software só que voltado para a *Web*, de maneira que se adapte e retifique o processo e cada prática aos atributos e características especiais das *WebApps* (PRESSMAN e LOWE, 2009).

Para diminuir os custos e problemas gerados por esses erros existe uma parte da Engenharia *Web* encarregada de fazer testes. A principal justificativa para elaborar um teste de uma aplicação é o seu alto custo de manutenção futuras devido a erros ou defeitos. Um estudo realizado pelo NIST – *National Institute of Standard sand Technology* aponta que nos Estados Unidos, os custos com erros de software podem chegar aproximadamente 60 bilhões de dólares por ano. O mesmo estudo afirma que, investimentos em testes de software poderiam gerar uma economia de aproximadamente 22 bilhões de dólares por ano. O

especialista em Testes de software, Rodrigo Zauza Passos, afirma que os testes podem ser realizados desde a concepção do software, por meio de avaliação de documentos, até a sua finalização, com a execução do sistema por técnicos especializados (COMPUTER WORLD, 2002).

São abordados neste trabalho algumas dimensões da área de teste descritas abaixo :

- o conteúdo que avalia a parte escrita, se há algum erro de ortografia ou violações de direitos autorais;
- a navegabilidade para garantir toda a sintaxe de navegação;
- o desempenho do site também se suporta grande número de usuários;
- a compatibilidade de diferentes navegadores;

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como finalidade fornecer um referencial teórico sobre conceitos de Engenharia de *Software* e Engenharia *Web*, tipos de testes para *Web* e ferramentas de testes.

2.1 ENGENHARIA DE SOFTWARE

O software tornou-se extremamente necessário em praticamente todos os aspectos da vida humana, o número de pessoas envolvidas pelos recursos fornecidos por tal aplicação vem crescendo constantemente. Ao elaborar uma aplicação seja ela *Web* ou não, muitas pessoas devem ser ouvidas, os requisitos de tecnologia de informação demandados por indivíduos, empresas e órgãos governamentais estão se tornando cada vez mais complexo a cada ano. O que antes era produzido somente por uma pessoa, hoje precisa de equipes grandes para seu desenvolvimento. Conforme o valor de uma aplicação aumenta, o esperado é que o número de usuários e a longevidade da aplicação aumentem. Com isso a demanda por manutenção, adaptação ou melhoramento também aumentam. Com essa simples verificação, conclui-se que um software em todas as suas formas e campos de aplicação devem passar pelos processos de engenharia (PRESSMAN, 2011).

Não pode-se falar sobre Engenharia *Web* sem antes falar sobre suas origens que é a Engenharia de software. Ela se preocupa com todos os aspectos de produção de um software, desde sua especificação inicial até a sua manutenção quando o sistema já esta sendo usado, tem por objetivo acompanhar o desenvolvimento profissional do software, inclui técnicas que apoiam especificação, projeto e evolução de programas que normalmente são relevantes para o desenvolvimento do software pessoal. (SOMMERVILLE, 2011).

Uma abordagem sistemática na Engenharia de software é às vezes chamada de processo onde o mesmo é uma sequência de atividades que levam a elaboração de um software. Normalmente essas etapas se resumem em realizar especificações, suas funcionalidades e restrições, o desenvolvimento, a validação e a evolução do software (SOMMERVILLE, 2011).

Não existem técnicas e métodos universais na Engenharia de software adequada a todos os sistemas e todas as empresas, mas sim um conjunto de métodos e ferramentas que vem evoluindo nos últimos 50 anos. De forma geral engenheiros *Web* aplicam a teoria, ferramentas e métodos sempre em busca de soluções para um problema, na engenharia o foco

não é somente com os processos técnicos de desenvolvimento de software, ela inclui atividades como gerenciamento de projeto, desenvolvimento de ferramentas, métodos e teoria para acompanhar a produção de um software. (SOMMERVILLE, 2011).

2.2 ENGENHARIA WEB

No início a Internet era apenas armazenamento de informações com acesso universal e tinha pouco ou quase nenhum efeito nos sistemas de softwares que eram acessados somente dentro de organizações. Por volta do ano 2000 a Internet começou a progredir, os navegadores também começaram a ter vários recursos e com isso tornando possível o desenvolvimento de softwares que passaram a ser acessados por navegadores. Isso levou ao desenvolvimento de enorme quantidade de novos produtos de softwares trazendo serviços inovadores acessados através da Internet. (SOMMERVILLE, 2011).

A *Web* se tornou indispensável para os negócios, comércio, comunicação, educação, engenharia, governo, finanças, medicina, política, ciência, transportes e muitos outros, sem falar que mudou até a forma como as pessoas conversam, fazem compras e ficam informados do que acontece no mundo.

A Engenharia Web incide sobre metodologias, técnicas e ferramentas que são baseadas no desenvolvimento de aplicações *Web* e que apoiem a sua concepção, desenvolvimento, evolução e avaliação. O desenvolvimento de aplicações *Web* tem características que tornam diferente do software convencional, é multidisciplinar e engloba contribuição de diversas áreas: sistemas, análise e design, hipermídia / hipertexto, engenharia de requisitos, interface do usuário, recuperação de dados, testes, modelagem e simulação, gerenciamento de projetos, engenharia de usabilidade, gráficos, projeto e apresentação. (ALKHATIB e RINE, 2010).

A Engenharia *Web* propõe uma forma de adequar o processo de sistemas consistidos na *Web* já baseados em toda a Engenharia de software compreendida só que com características específicas que possam ajudar o desenvolvedor a extrair todos os requisitos do cliente e elaborar um sistema muito mais eficiente e tornar a sua produção mais rápida. Essa engenharia propõe que o desenvolvedor tenha uma visão mais aberta e dinâmica e conseqüentemente ser capaz de elaborar um sistema muito mais confiável, seguro e com qualidade.

Uma empresa moderna exige adaptação, pois estratégias de negócios e regras mudam rapidamente, a gerência exige respostas quase instantaneamente, essa deve ser a visão de um engenheiro *Web*, conseguir acompanhar essa velocidade. Um cliente se interessa por uma

WebApp quando ela é entregue e não pelo trabalho envolvido na criação de uma *WebApp* (PRESSMAN e LOWE, 2009).

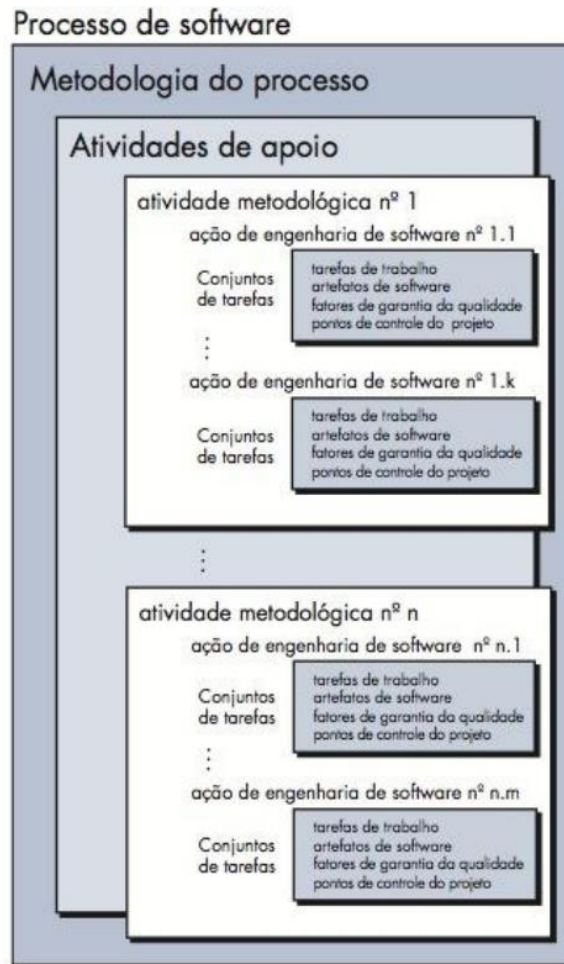


Figura 1: Processo de Engenharia Web
Fonte: PRESSMAN (2011)

Um arcabouço estabelece a base para um processo completo de Engenharia *Web*, se resume em um conjunto de tarefas que produz um produto, cada ação é preenchida com tarefas individuais, que executam alguma parte do trabalho conforme pode ser observada na Figura 1.

Segundo Pressman (2006), as atividades a seguir fazem parte de um arcabouço genérico e é aplicada a grande maioria dos projetos *Web*:

- **Comunicação:** Engloba a comunicação com o cliente e outros interessados abrangendo o levantamento de requisitos e atividades relacionadas;

- **Planejamento:** Estabelece as ações da Engenharia *Web* que ocorrerão, as tarefas e técnicas a serem utilizadas, os riscos prováveis, os recursos que serão exigidos, os produtos de trabalho a serem produzidos e um cronograma a ser seguido;
- **Modelagem:** Envolve a criação de modelos que auxiliam o desenvolvedor e o cliente a entenderem melhor os requisitos da *WebApp* e o projeto que satisfará esse requisito;
- **Construção:** Esta parte é a programação em si, envolve a combinação de HTML, XML, PHP, ou código semelhante, juntamente com o teste necessário para revelar erros no código;
- **Implantação:** Entrega um incremento da *WebApp* ao cliente, que avalia e oferece com base na avaliação;

Essas cinco atividades podem ser utilizadas durante todo o processo de desenvolvimento de uma aplicação *Web* seja ela o tamanho e a complexidade de for, os detalhes específicos de cada projeto se diferenciam, mas o “esqueleto” principal continua sendo o mesmo.

2.2.1 Um projeto *WebApp*

Um projeto *WebApp* aborda técnicas onde é assimilada a clareza e a fisionomia de um site onde o *layout* é criado, e definido a estrutura da arquitetura e uma estrutura de navegação.

Nielsen apud PRESSMAN (2009) afirma que existem basicamente duas abordagens básicas para um projeto *WebApp*: Um ideal artístico de se expressar e o ideal da engenharia de se resolver um problema para um cliente. Nos primeiros anos de desenvolvimento *Web* o lado artístico foi levado mais em conta na medida em que o HTML – *Hyper Text Markup Language* foi sendo desenvolvido.

Conforme Kaiser apud PRESSMAN (2009) algumas considerações devem ser levados para a construção de praticamente qualquer *WebApp*, seja qual for o seu propósito, tamanho ou complexidade:

- **Simplicidade:** Tudo em excesso não é bom, para uma página da Internet também vale a regra, uma *WebApp* não deve conter conteúdo extenuante, páginas enormes devem ser mantidas na simplicidade e moderado;
- **Consistência:** Nesse contexto, toda a formatação de texto e estilo de fonte devem ser a mesma, arte gráfica deve ter tons que combinem, o projeto da arquitetura deve ser uniforme para uma fácil navegação e acesso a estrutura de hipermídia;

- **Identidade:** A estética, interface e projeto de navegação devem ser projetados de acordo com o domínio de aplicação para qual será construído, por exemplo, um site de comércio eletrônico é bem diferente de um site financeiro. O projeto é construído levando em conta diferentes categorias de usuários, organizado para cumprir objetivos diferentes;
- **Robustez:** O usuário deseja que a função esperada do aplicativo funcione, se algum elemento que foi implicado nos requisitos faltar ou falhar, este aplicativo deixa de ser robusto;
- **Navegabilidade:** Além da simplicidade, um projeto *WebApp* deve ser claro e previsível, o usuário deve entender como navegar pelo site sem ficar procurando instruções e links para navegação;
- **Apelo Visual:** A beleza de um projeto *WebApp* é um conceito que varia segundo a ótica de quem a vê, tudo contribui para um projeto bem feito ou não, como mencionado no item Simplicidade, dependendo do nicho deve-se ter cuidado com o *layout* projetado;
- **Compatibilidade:** Para a construção de uma *WebApp* deve-se levar em conta que o mesmo será usado em vários ambientes (navegadores diferentes, plataformas diferentes, sistemas operacionais, conexão com Internet) onde deve ser compatível com cada um deles;

2.2.2 Projeto de interface

A interface de um projeto independente do conteúdo ou serviço que é prestado é sua referência ao usuário e deixa sempre uma primeira impressão. Uma interface que é mal produzida pode deixar o usuário em dúvida quanto sua integridade. Uma interface deve “atrair” o usuário.

Segundo Tognozzi apud PRESSMAN (2011), uma *WebApp* deve ser projetada para prever o próximo passo do usuário, por exemplo, quando um usuário solicita um conteúdo que apresenta informações sobre um driver de um computador. Neste exemplo o sistema deve prever que esse usuário queira fazer o download desses *drivers* e apresentar formas para o usuário fazer isso sem precisar buscar recursos. Uma *WebApp* deve ser flexível, o usuário deve compreender aonde está e também lhe fornecer recursos para desfazer erros e refazer caminhos de navegação mal escolhidos.

Ao carregar uma *WebApp* ela deve ser dinâmica, ela deve usar outros recursos enquanto uma imagem é carregada, assim o usuário pode continuar com a navegação como se a operação tivesse sido realizada, e também a *WebApp* deve fornecer algo que mostre ao usuário o que estiver carregando ainda esta em processo de download, por exemplo, uma barra de 0 a 100% para o usuário não ter uma impressão que a *WebApp* está em processo ou “travada”.

Uma interface *WebApp* deve ser projetada pensando na redução do tempo em que o usuário leva para a aprendizagem, deve ser simples e intuitivo e o conteúdo deve ser organizado para que tudo fique óbvio ao usuário.

Segundo Nielsen e Wagner Apud Pressman (2011) com experiência em reprojeto de uma importante *WebApp* sugere que:

- Partes do texto devem ser excluídas, isso porque ler rápido em um monitor de computador é aproximadamente 25% mais lento que ler um papel, logo se torna cansativo;
- Evitar sinais “em construção” pois é um link desnecessário;
- Usuários preferem “rolar” a página, informações relevantes devem ser posicionadas em uma típica janela de navegação;
- As opções de navegação devem ser óbvias. Um usuário não deve ficar procurando na tela um link que procura ou conteúdo que deseja explorar;
- Em uma visão geral, o projeto de *layout* deve ser bem projetado para aumentar a percepção do usuário do conteúdo ou serviço prestado, não precisa ser chamativo e sim bem consolidada. A seguir é descrito as tarefas para um fluxo de trabalho para um projeto *WebApp*:
- Revisar todas as informações contidas nos requisitos do usuário e ajustá-los conforme necessidade;
- Desenvolver um esboço do *layout* para uma interface;

Esta etapa é muito importante para o restante do desenvolvimento, pois o cliente pode ter uma noção de como vai ficar a *WebApp* e se houver algo em que não ficou bem claro na atividade de requisitos, nesta etapa pode ser corrigida visto que é somente um esboço.

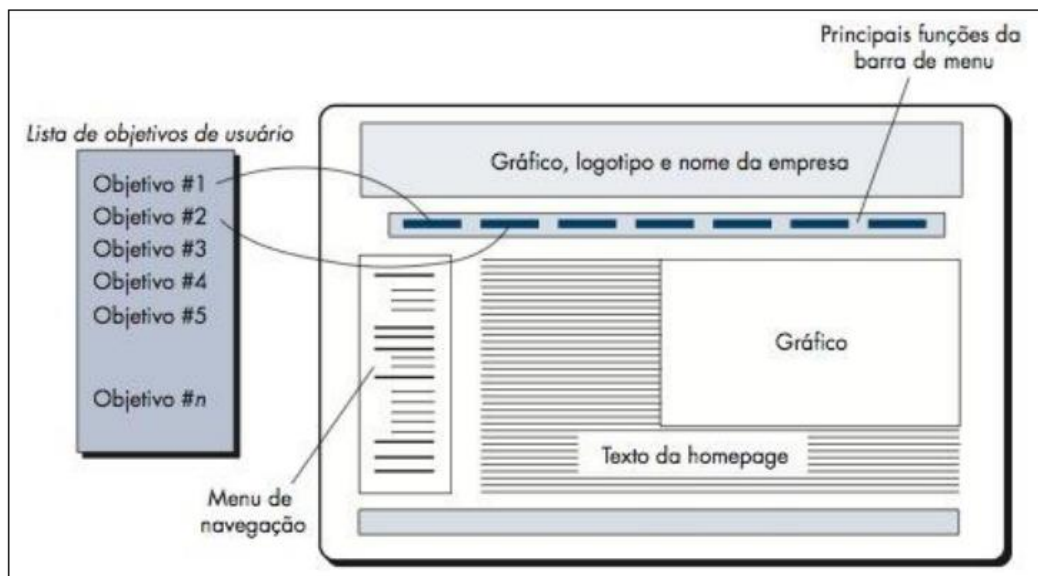


Figura 2: Projeto padrão
Fonte: PRESSMAN (2011)

- Deixar bem a mostra os objetivos primários dos usuários da *WebApp*, na Figura 2 pode-se observar os objetivos bem explícitos, assim o usuário localiza melhor o seu objetivo;
- Definir um conjunto de tarefas a ser seguida pelo objetivo do usuário, por exemplo em um site de compras, a opção comprar produto vem associado a formas de pagamento, carrinho de compras, preços, etc;
- Elaborar imagens de tela para representar cada interação com o usuário, mostrando as funcionalidades da *WebApp* e indicar os links de navegação;
- Revisar o modelo de projeto inicial se concentrando na usabilidade;

Ao final do protótipo operacional, ele deve ser avaliado para ver se atende aos requisitos do usuário, na Figura 3 é ilustrado um ciclo de avaliação para interface do usuário, pode ser elaborado um teste informal ou até mesmo um questionário formal que é avaliado por um número x de usuários, podem-se extrair resultados do questionário, por exemplo, 70% de todos os usuários que preencheram o questionário não gostaram da forma como são apresentados os dados de tal página. São Realizadas modificações necessárias baseado nas informações dos usuários e um novo projeto é apresentado, o ciclo termina até que não apresente mais modificações solicitadas pelos usuários.

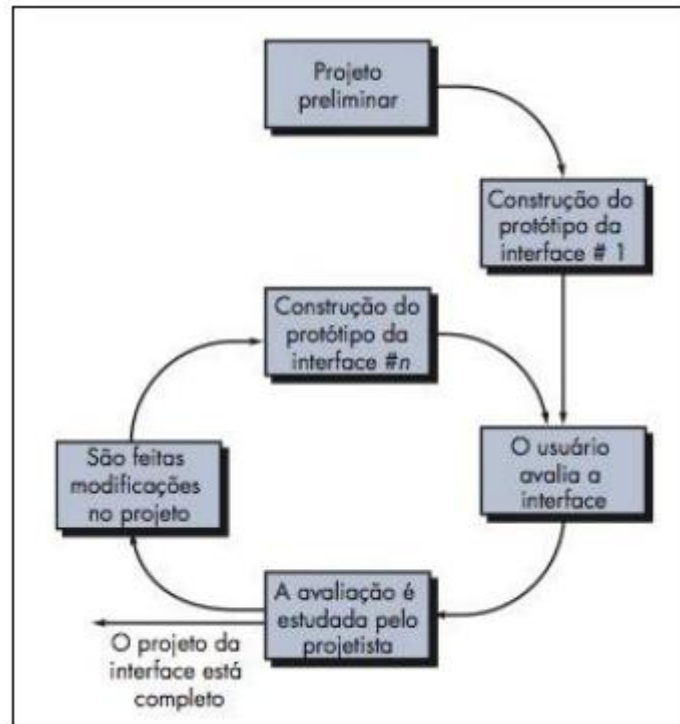


Figura 3: Ciclo de avaliação
Fonte: PRESSMAN (2011)

2.2.3 Qualidade de software

Todos os usuários que navegam pela Internet acabam acessando diversos tipos de sites, com diversos tipos de conteúdos e todos tem sua opinião formada do que faz uma *WebApp* ser “bom”, dentre eles diversos tipos de gostos, entre eles aqueles que gostam de *WebApp* relacionado a música, entretenimento, notícias, compras, que contenham muito ou pouco texto, simplicidade ou ferramentas sofisticadas.

Em 2005, a *Computer World* publicou que software de má qualidade está presente em quase todas as organizações que fazem uso de computadores provocando muito tempo perdido durante o tempo em que a máquina fica parada, dados corrompidos, oportunidades de vendas perdidas, custos com suporte, manutenção de TI altos e baixa satisfação do cliente (PRESSMAN, 2011).

Qualidade de software se define pela visão do usuário, um software ou uma *WebApp* que atenda os requisitos que ele solicitou. Pela visão do fabricante se atende às especificações originais do produto e pela visão do produto sugere-se a qualidade ligada a suas funções e recursos estão em completo funcionamento

No desenvolvimento, a qualidade de um projeto abrange o grau de atendimento às funções e característica específica no modelo de requisitos, ela é atingida quando a

implementação segue o projeto e o resultado atende as necessidades e as metas de desempenho (Hildreth apud PRESSMAN, 2011).

Offutt apud PRESSMAN (2011) foca cinco principais atributos de qualidade que são apresentados na Figura 4:

- **Segurança:** A partir do momento em que uma *WebApp* é integrado a um banco de dados ela precisa necessariamente ser segura para atender um requisito de qualidade, principalmente em situações em que é armazenado informações confidenciais de clientes como é o caso de bancos online e comércio eletrônico.
- **Disponibilidade:** Pode ser a melhor *WebApp* para acesso de conteúdo ou segurança, mas não terá qualidade se a mesma não estiver disponível, para um cliente é inaceitável uma *WebApp* não estar acessível 24h/dia.
- **Escalabilidade:** O servidor dessa *WebApp* deve atender a quantidade de usuários que foi designada, está ligado diretamente com a disponibilidade, se um número relativamente alto de usuários tentar acessar a *WebApp* a mesma deve ter suporte para esse número de acesso.
- **Tempo de colocação no mercado:** Não é um atributo ligado diretamente com a *WebApp*, mas sendo a primeira a entrar no mercado captura um número de usuários bem maior.

A qualidade é incorporada a uma aplicação *Web* como consequência de um bom projeto

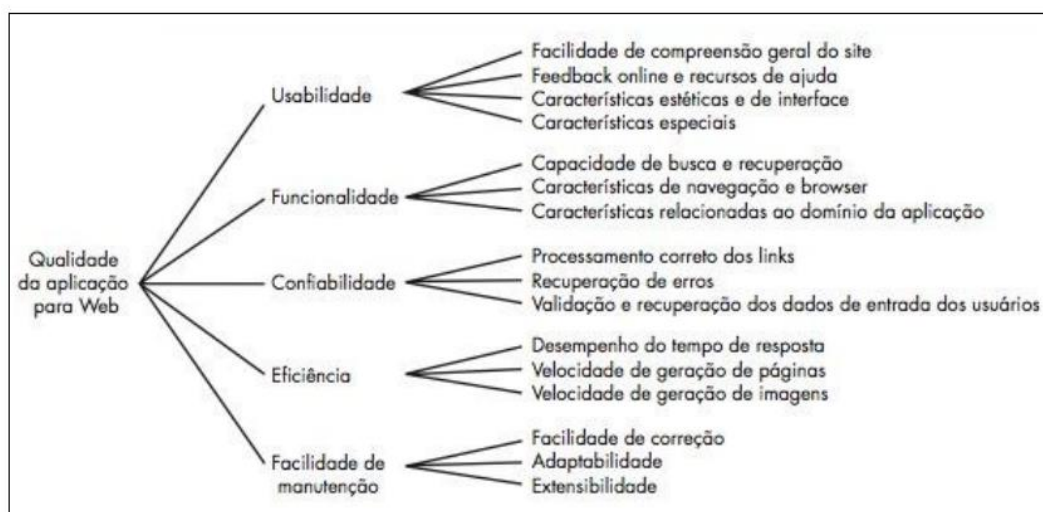


Figura 4: Atributos de qualidade
Fonte: PRESSMAN (2011)

A garantia de qualidade de software, SQA – *software Quality Assurance* é uma atividade universal que é aplicada em toda a gestão de qualidade. Sua garantia engloba um processo de SQA, tarefas específicas de garantia da qualidade e controle da qualidade, práticas de Engenharia de software, controle de artefatos e mudanças feitas, um procedimento para garantir a conformidade com os padrões de desenvolvimento de software e medições e relatórios (PRESSMAN, 2011).

A seguir é descrito um conjunto de metas a serem realizados pela SQA:

- **Qualidade de requisitos:** A SQA deve assegurar que todos os requisitos do usuário foram vistos e revistos para garantir um alto nível de qualidade, verificar atributos como a ambiguidade, número de mudanças nos requisitos, clareza do modelo;
- **Qualidade de projeto:** Todo o modelo de projeto deve ser avaliado pela equipe para garantir que esteja de acordo com os requisitos, a SQA busca atributos para garantir a qualidade do projeto como a complexidade dos componentes, avalia por exemplo o número de cliques para chega a tal conteúdo;
- **Qualidade de código:** Todo o código-fonte deve estar em conformidade com os padrões locais de codificação e que são projetados para facilitar a manutenção, é analisado nessa meta o número de padrões usado no código;
- **Eficácia do controle de qualidade:** A SQA analisa a aplicação dos recursos da equipe de desenvolvimento para assegurar que eles estão sendo alocados de maneira efetiva. Mesmo que não haja uma equipe de SQA, pode-se seguir um plano de tarefas pela equipe de software, assim garantindo a qualidade do mesmo;

2.2.4 Custo de qualidade

A qualidade conta muito para um sucesso de uma *WebApp*, ela custa tempo e dinheiro, a sua falta custa dinheiro tanto para o cliente final que terá que se deparar com o erro quanto para a entidade que criou a *WebApp* que terá que fazer manutenção. O custo de uma aplicação *Web* depende de quanto de qualidade a mesma terá, este custo inclui todos os recursos necessários para execução de atividades que eliminam a falta de qualidade, reunindo métricas

para prover uma base de custos, estes custos podem ser divididos em prevenção, avaliação e falhas.

- **Custo de prevenção:** Inclui planejamento para atividades de gerenciamento necessário, testes e treinamento relacionado a *WebApp*;
- **Custo para avaliação:** Inclui revisões técnicas, coleta de dados e avaliação de métricas e depurações;
- **Custo para falhas:** Este item desapareceria caso a *WebApp* não apresenta-se erros antes ou depois da entrega do produto final, que podem ser divididos em custos de falhas internas (erros antes da entrega) que engloba retrabalhos para correção de erros, efeitos colaterais gerados devido a mudança de código e falhas externos (após entrega ao cliente) que inclui resoluções de reclamações, devolução ou substituição, suporte via telefone/email, mão de obra associado a garantia do produto, má reputação e como consequência perda de negócios;

À medida que se aumentam as previsões de custos com prevenções e detecção de falhas esses valores pode aumentar consideravelmente, na Figura 5 pode-se verificar os custos de um projeto, que varia de média, a economia que pode ser obtida associada a atividades iniciais de controle com garantia de qualidade é considerável



Figura 5: Custos de um projeto
Fonte: PRESSMAN (2011)

2.3 TESTES PARA APLICAÇÃO WEB

Conforme estudo conduzido pelo NIST –*National Institute Of Standards and Technology*, os defeitos resultam em um custo anual aproximado de 60 bilhões de dólares a economia dos Estados Unidos, mais de um terço do custo poderia ser evitada com melhorias na infraestrutura do teste de software.(COMPUTERWORLD, 2002).

Segundo Mendes e Mosley (2006) as aplicações de baixa qualidade são difíceis de manter e que os principais problemas estão no design e o desenvolvimento inadequado nos processos e práticas de gerenciamento de projeto. Seguem alguns problemas com terceirização de projetos *Web*:

- 84% dos projetos entregues pesquisados não atender às necessidades de negócios;
- 53% dos projetos entregues pesquisados não forneceu a necessária funcionalidade;
- 79% dos projetos pesquisados apresentaram atrasos no cronograma;
- 63 % dos projetos pesquisados excedeu seu orçamento;

Para reduzir prejuízos causados por sistemas *Web*, existe uma parte da Engenharia *Web* que trata esses problemas, que é testes em softwares *Web*. Basicamente os testes consistem em executar o programa ou rodar uma página com intenção de encontrar erros. Um processo de desenvolvimento que adie o teste, até que todos os componentes possam ser montados em um sistema completo, é uma proposição arriscada. Defeitos encontrados tardiamente no ciclo de vida serão mais difíceis de consertar e provavelmente provocarão atrasos na programação, principalmente se forem problemas arquiteturais que exijam um novo projeto para serem corrigidos (MYERS, 2004).

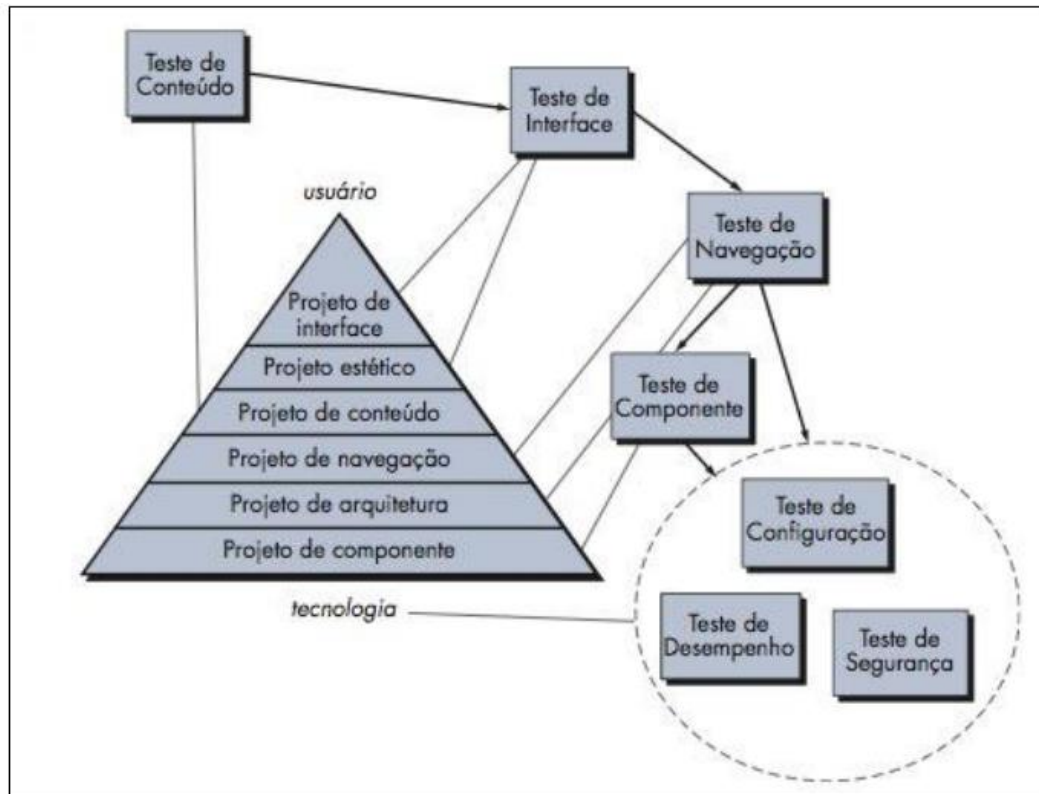
Segundo PRESSMAN (2011) deve-se levar em consideração as dimensões da qualidade de uma *WebApp*, a qualidade é incorporada a uma aplicação *Web* como consequência de um bom projeto, a seguir é descrito algumas dimensões que são relevantes em qualquer discussão de testes da *WebApp*:

- **Conteúdo:** São avaliadas questões de sintaxe, grafia, pontuação e gramática da documentação baseada em textos e também a falta de ambiguidade;
- **Função:** É testado para garantir que os requisitos do cliente estão sendo atendido. Cada função é verificado quanto a exatidão, instabilidade e conformidade geral com os padrões de implantação apropriados;
- **Estrutura:** É avaliado para garantir corretamente o conteúdo e função da *WebApp*, que pode ter suporte para novo conteúdo ou funcionalidade;

- **Usabilidade:** Testado para garantir cada categoria de usuário tenha suporte da interface para que o cliente possa aprender e manusear todo o conteúdo do site;
- **Desempenho:** É avaliado sob uma variedade condições de operação, configuração e carga para garantir que o *WebApp* atente as expectativas sem uma degradação operacional inaceitável;
- **Compatibilidade:** É testado executando o *WebApp* em várias plataformas diferentes, buscando encontrar erros em alguma específica;
- **Interoperabilidade:** Testado para garantir que a *WebApp* realiza a interface correta com outras aplicações ou banco de dados;
- **Segurança:** É testado para avaliar grandes vulnerabilidades tentando explorá-las o máximo possível. Qualquer penetração bem sucedida é considerada uma falha de segurança;

2.3.1 Visão Geral

O processo de teste para Engenharia *Web* de uma visão geral engloba praticamente todas as etapas do processo de desenvolvimento de uma *WebApp*. Na Figura 6 uma pirâmide de projeto *WebApp* é construída, primeiramente é testado e revisto o conteúdo, o fluxo de teste ocorre da esquerda para a direita e de cima para baixo, o que é visível para o usuário é o que fica no topo da pirâmide, seguindo pelos processos da estrutura, questões de instalação e implantação da *WebApp*.



6: Pirâmide de construção
Fonte: PRESSMAN (2011)

2.3.2 Teste de conteúdo

Nessa etapa, é revisto todo o conteúdo como uma revisão de documento escrito, esses erros podem ser desde pequenos erros tipográficos até informações incorretas, violação de lei e imagens sem direitos autorais.

O teste de conteúdo busca descobrir erros de sintaxe (erros ortográficos e gramaticais) em um documento de texto, representações gráficas e outros meios, descobrir erros na exatidão ou integridade das informações em qualquer objeto que é visto na navegação e encontrar erros na estrutura ou organização do conteúdo fornecido para o usuário final.

O teste semântico pode ser seguido alguns requisitos que segue abaixo:

- Informação atualizada e com base em fatos;
- Informação concisa de direta ao ponto;
- Leiaute fácil de entender;
- Objetos que estão dentro de um conteúdo são encontrados facilmente ou consiste com informações apresentado em outro objeto de conteúdo;

- Conteúdo ofensivo ou confuso;
- Desrespeita direitos autorais ou marcas registradas;
- Links que completam o conteúdo estão ancorados corretamente;
- Estética do conteúdo condiz com a estética da interface;

Conseguir testar todos esses requisitos pode ser um tanto complicado, mas é necessário para assegurar a qualidade de uma *WebApp*, será muito desapontador para um cliente visitar uma dessas *WebApp* e pesquisar e ler sobre um conteúdo X e a imagem ser de um conteúdo Y.

WebApps modernas normalmente não apresentam o conteúdo somente estático, existem muitos casos em que o conteúdo é atualizado em tempo real, faz uso de conexão com o banco de dados. Um exemplo uma *WebApp* que apresenta notícias de uma certa região, neste caso, o cuidado para não colocar um conteúdo errado fica a parte da organização que faz a atualização e não dos desenvolvedores que deram o treinamento adequado. Os erros podem ocorrer por uma solicitação de informação do lado do cliente que raramente pode ser colocado em um gerenciamento de banco de dados, deverão ser apresentados testes para descobrir esses erros. (PRESSMAN e LOWE, 2009).

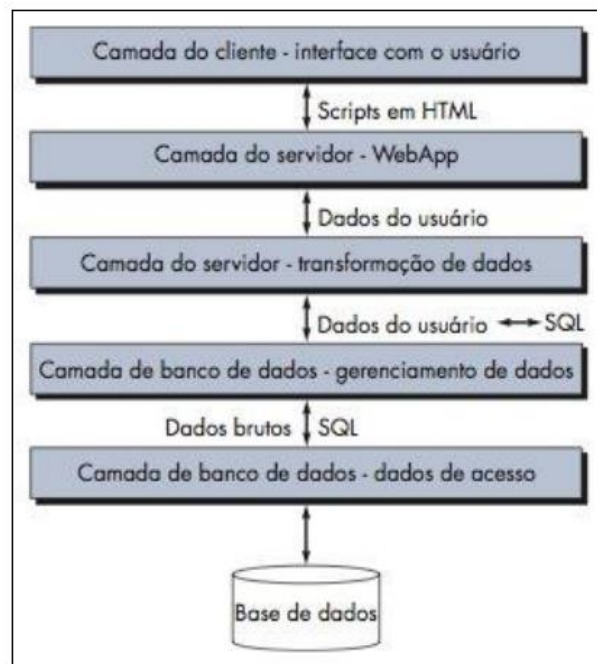


Figura 7: Camadas entre cliente e servidor
Fonte: Fonte: PRESSMAN (2011)

Devem ser desenvolvidos testes que descubram erros na comunicação entre *WebApp* e o banco de dados. Na Figura 7 é possível visualizar as diversas camadas existentes entre o cliente e a base de dados, onde todas essas etapas devem prever possíveis erros. Deve haver um processo de teste para garantir que os *scripts* sejam construídos corretamente para cada consulta de usuário e transmitidos adequadamente para o lado do servidor. Na sequência a camada de *WebApp* do lado do servidor é testada para assegurar que os dados do usuário sejam extraídos e transmitidos corretamente para a camada de transformação, as funções dessa transformação devem ser testados para assegurar que o SQL correto seja criado e passados para os componentes apropriados de gerenciamento de dados.(PRESSMAN,2011)

2.3.3 Teste de interface de Usuário

Os testes de interface começam desde a análise, que é revisado para garantir que esteja em conformidade com os requisitos dos interessados, incluindo análises do caso de uso. Durante o projeto é estabelecido critérios genéricos de qualidade para todas as interfaces de usuário. O próximo passo são os testes após o término do projeto, que experimentam mecanismos de interação e valida os aspectos estéticos da interface do usuário. O objetivo é encontrar erros relacionados com mecanismos da interface, na maneira como é implementada as semânticas de navegação, funcionalidades da *WebApp* e exibição de conteúdo.

Esteticamente é avaliada características como uso da cor, molduras, imagens, bordas, tabelas e características de interface. São projetados testes para experimentar formulários, *scripts*, HTML dinâmico, conteúdo concatenado (exemplo um carrinho de compras em uma *WebApp* de vendas). À medida que o usuário interage com uma *WebApp*, ele se depara com mais de um mecanismo de interface, abaixo são descrito algumas considerações de interface:

- **Links:** É elaborada uma lista com todos os links a serem testados, incluindo menus, índice e outros. É executado um teste em cada um desses itens verificando se o objeto de conteúdo ou a função apropriada sejam acessados, também são testados links externos e avaliado o risco para se tornarem indisponível com o tempo;
- **Formulários:** São verificados os campos, se é realmente o campo do título e se os campos obrigatórios são visualmente identificados para o usuário, se as máscaras condizem com o campo, por exemplo, campos onde só aceitam

números não poder aceitar letras, o preenchimento automático também deve estar correto para que a tecla *tab* faça a transição de um campo para o outro de forma apropriado. É testado se o servidor recebe todas as informações sem perda de dados e se os *scripts* que verificam erros sobre dados introduzidos funcionem corretamente e enviem mensagem de erro coerente;

- **HTML dinâmico:** Cada página que conter uma HTML dinâmico é executada para assegurar que a mesma funcione, também é testada em ambientes que suportam as *WebApp*;
- **Janelas pop-up:** São realizados testes para verificar se estão corretamente dimensionadas, se o projeto estético está consistente com o da interface e todas as funcionalidades como barra de rolagem.
- **Conteúdo encadeado:** Os testes devem demonstrar que os dados encadeados são atualizados, exibidos, suspensos e reiniciados sem dificuldade;
- **Cookies:** No lado do servidor deve verificar se o *cookie* foi corretamente construído e transmitido para o lado do cliente. No lado do cliente é verificado se a *WebApp* anexa corretamente os *cookies* existentes de acordo com a solicitação;

Após analisar cada um desses itens, o teste segue para a semântica de interface. Este teste avalia o quanto o projeto se preocupa com o usuário, se oferece diretrizes claras e verificação da correta consistência de linguagem e abordagem. É aconselhável manter um *checklist* para garantir que todas as etapas foram concluídas e pelo menos todos os links contidos em um objeto tenham sido utilizados (PRESSMAN,2011).

2.3.4 Teste de Usabilidade

Os teste de usabilidade são parecidos com o teste de interface de usuário pois também avaliam o grau de interação que o usuário tem com a *WebApp*. Basicamente o teste determina o grau com que a *WebApp* facilita a vida do usuário. Pode-se usar o seguinte método para o teste:

- Definir um conjunto de categorias de teste de usabilidade e identificar objetivos para cada um;
- Elaborar testes para avaliar cada objeto;

Selecionar um grupo de pessoas para realizar os testes;

- Registrar os detalhes da interação dos participantes enquanto realizam os testes;
- Desenvolver um mecanismo para avaliar a usabilidade da *WebApp*;
- Os testes podem ter focos em várias categorias, pode-se investigar um mecanismo específico como um formulário por exemplo;

Testes com usuários tem como objetivo investigar o comportamento dos usuários reais, observado através de um amostra representativa de usuários reais. Ela exige que os usuários executem um conjunto de tarefas usando artefatos físicos, que pode ser protótipos ou aplicações prontas, enquanto um investigador observa seu comportamento e reúne dados sobre a forma como os usuários executam tarefas atribuídas. Em geral, os dados recolhidos durante as investigações são o tempo de execução do usuário, número de erros e satisfação do usuário. Após o teste de usuário estiver concluída, os dados recolhidos são analisados e utilizados para melhorar a usabilidade do aplicativo. O teste de usabilidade é explicitamente dedicado a analisar em detalhe como os usuários interagem com o aplicativo ao realizar tarefas bem definidas (MENDES e MOSLEY, 2006).

A seguir são descritas algumas categorias que devem ser consideradas durante a realização dos testes:

- **Interatividade** – Verificar se os menus, botões e ponteiros são fáceis de usar;
- **Layout** – Verificar se os mecanismos de busca, conteúdo e funções são alocados de uma maneira que o usuário encontre rapidamente;
- **Clareza** – Avaliar se os textos são bem escritos e fácil de entender, incluindo as imagens;
- **Estética** – O usuário deve se sentir confortável ao acessar a página, o *layout*, tipo de letra e cores devem estar coerentes;
- **Características de tela** – Verificar se a *WebApp* otimiza o tamanho e resolução da tela;
- **Personalização** – Observar se a *WebApp* se adapta as necessidades específicas de diferentes usuários;
- **Acessibilidade** – Suporte para usuários com necessidades especiais;

Para cada categoria é realizado uma série de teste, alguns casos somente uma revisão visual atente as necessidades. Algumas características devem ser avaliadas como: animação, botões, cores, controle, diálogo, campos, formulários, molduras, gráficos, rótulos, *links*, menus, mensagens, páginas de navegação, seletores, texto e barras de ferramentas. A Figura 8 apresenta um conjunto de “graus” de avaliação que podem ser optados pelos usuários, podendo ser aplicado em cada característica individualmente para uma página *Web* ou para a *WebApp* completa.(PRESSMAN,2011)



Figura 8: Conjunto de graus
Fonte: PRESSMAN (2011)

2.3.5 Teste de Compatibilidade

É certo que uma *WebApp* seja acessada de diversas formas que existem, entre elas pode-se citar: navegadores diferentes onde uma página acessada pode apresentar pequenas diferenças independente do grau de padronização do HTML ou também podem alterar significativamente o *layout* da página, computadores diferentes que podem resultar diferenças na velocidade, sistemas operacionais, velocidade de conexão de rede, os *plug-ins* podem estar faltando tornando o conteúdo indisponível, estilos de fonte podem ser alterados tornando difícil a leitura do mesmo, formulários podem ser montados de forma errada. O teste de compatibilidade tenta descobrir esses erros antes que a *WebApp* torne-se online.

Primeiramente deve-se definir uma série de configurações de computadores disponíveis no lado do cliente, identificando cada plataforma, dispositivo, sistemas operacionais, navegadores disponíveis, velocidades prováveis de conexão e informações que correspondam a esses itens. Em seguida deve haver uma série de testes de validação de compatibilidade, muitas vezes adaptado ao teste de interface, navegação e desempenho com finalidade de encontrar possíveis diferenças de configuração. (PRESSMAN, 2011)

Existe a técnica de teste de erro forçado que é usado para gerar casos de testes que propositalmente conduzem o componente de uma *WebApp* a dar erro, a finalidade é descobrir erros que ocorram durante a manifestação do erro para averiguar as mensagens, falhas, saída e

entrada errônea, por exemplo, em uma *WebApp* de um *e-commerce*, o número de dias para entrega que é de 2 a 15 dias, o teste deve ser efetuado com valores fora dessa margem para verificar que a página está dando a mensagem de erro que foi projetado (NGUYEN apud PRESSMAN, 2011).

2.3.6 Teste de Navegação

Nenhum usuário tem a visão igual a de outro, sempre ao entrar em uma *WebApp* são tomados caminhos diferentes, aprendem ou iniciam atividades e tomam decisões, tudo é imprevisível, ele pode escolher um caminho ou iniciar uma ação. A tarefa do teste de navegação é garantir que todos os mecanismos que um usuário pode navegar estejam em completo funcionamento e confirmar que cada semântica de navegação possa ser alcançada pela categoria de usuário. (PRESSMAN, 2011).

A primeira etapa do teste de navegação inicia durante o teste de interface. Splaine apud Pressman (2011) sugere os seguintes mecanismos de navegação a ser testados:

- **Links de navegação:** Incluem links internos, links externos para outra *WebApp* e links ancorados dentro de uma página específica, todos esses links devem ser testados para assegurar o seu funcionamento;
- **Redirecionamentos:** Quando o usuário solicita uma URL que não existe ou foi removido, é exibida uma mensagem para o usuário e a navegação é redirecionada para outra página que pode ser a página principal;
- **Mapas de site:** O mapa de site fornece uma tabela com o conteúdo inteiro da *WebApp* para cada página, cada entrada deverá ser testada para garantir que o link é direcionado para a página correta;
- **Dispositivos de busca interna:** Uma *WebApp* complexa contém muitos objetos de conteúdo, o teste de busca procura validar a precisão e totalidade da busca, se há erros na busca e recursos avançados de busca como filtros.

2.3.7 Teste de Configuração

Existe um grande fator do lado do cliente que é a variação e instabilidade de configurações disponíveis, como o hardware, sistemas operacionais, navegadores, capacidade

de armazenamento, velocidades de comunicação de rede e outros fatores que são difíceis de prever, além de mudanças de configuração de um usuário como atualização do sistema operacional, mudança ou atualização de um navegadores, nova velocidade de conexão. Dois usuários que não partilham da mesma configuração podem ter resultados diferentes, o teste de configuração não visa exercitar todas as configurações existentes, mesmo porque seria uma tarefa um tanto impossível, e sim testar um conjunto provável de configuração do cliente e do servidor para assegurar que as navegações de todos sejam a mesma e isolar possíveis erros.(PRESSMAN, 2011).

Segundo Pressman (2011) no lado do servidor são verificados se podem suportar as *WebApps* sem erro, incluindo servidores de banco de dados, sistema operacional, *firewall*. Alguns pontos são importantes e devem ser verificados como:

- A *WebApp* e o sistema operacional são compatíveis;
- Os arquivos, diretórios e dados de sistemas são criados quando a *WebApp* está operacional;
- *Firewalls* ou criptografia do servidor interfere no funcionamento ou velocidade da *WebApp*;
- A *WebApp* foi testada com configuração de distribuidor distribuído (se for o caso);
- A *WebApp* está adequadamente integrada com o software de banco de dados e sensível a suas atualizações;
- Os *scripts* estão sendo executados corretamente;
- Erros de administradores podem afetar na *WebApp* até que ponto;

No lado do cliente, o teste de configuração focaliza na compatibilidade da *WebApp* com configurações que contenham diferenças nos seguintes componentes:

- *Hardware*: CPU, memória armazenamento e dispositivos de impressão;
- Sistemas operacionais: *Linux*, *Microsoft Windows*, OS, sistema móvel;
- *Software* navegador: Firefox, Internet Explorer, Safira, Google Chrome e outros;
- Componentes da interface de usuário: Active X, Java applets e outros;
- *Plug-ins*: *QuickTime*, *RealPlayer*, *FlashPlayer* e outros;
- Conectividade – Cabos, DSL, modem, TI, *Wifi*;

Em um projeto de testes de configuração do cliente, deve-se reduzir o número de variáveis de configuração a um número aceitável levando em consideração o tempo que lhe foi disponível. Cada categoria de usuário é avaliada para determinar as prováveis configurações que serão encontradas, pode-se utilizar dados de mercado para prever as configurações (PRESSMAN, 2011).

2.3.8 Teste de Segurança

O teste de segurança focaliza em três elementos distintos que são basicamente o lado do servidor que disponibiliza a *WebApp*, o caminho de comunicação entre o servidor e o cliente, e o ambiente do lado do cliente. Os testes focaliza o acesso não autorizado ao conteúdo e funcionalidades da *WebApp* que juntamente com o servidor e o ambiente que elas abrigam representam um alvo forte para *hackers* externos e outras pessoas que queiram ter acesso para trazer alguma mal ao funcionamento da *WebApp* roubando informações, modificando conteúdos, degradar o desempenho, desativar funcionalidades entre outras coisas. A organização responsável fica a tarefa de garantir a segurança e descobrir os pontos fracos que podem ser explorados por pessoas que pretendem fazer mal uso de informações (PRESSMAN e LOWE, 2009).

Um defeito comum é o estouro de *buffer* que permite que um código malicioso seja executado na máquina do cliente, por exemplo, introduzir uma URL maior do que o tamanho do *buffer* do navegador ira causar um erro de sobrescrita, se o navegador não tiver código para detecção de erro para validar o tamanho de entrada um *hacker* experiente pode aproveitar para escrever uma longa URL com código a ser executado para corromper dados, fazer travar o navegador entre outros. Outra vulnerabilidade do lado do cliente é o acesso não autorizado a *cookies* de navegadores, sites maliciosos podem usar informações de *cookies* para roubo de identidade.

Os dados transferidos entre servidor e cliente são vulneráveis, pode ocorrer uma enganação quando uma extremidade do elo é subvertida por uma entidade com intenções maliciosas que podem roubar senhas, informações confidenciais ou dados de crédito, por um simples site malicioso que pedem essas informações.

No lado do servidor pode haver ataques que causam recusa de atualizações e passa a ser vulnerável a *scripts* que podem passar para o lado do cliente ou desabilitar operações.

Algumas seguranças devem ser observadas em qualquer projeto *WebApp* (PRESSMAN, 2011).

- **Firewal:** Mecanismo de filtragem que combinam hardware e software que examinam pacotes de informação para averiguar se esta vindo de uma fonte legítima, bloqueando dados suspeitos;
- **Autenticação:** Valida a identidade de todos os clientes e servidores, permitindo que a comunicação ocorra somente quando ambos os lados são verificados;
- **Criptografia:** Mecanismos que modificam os dados confidenciais de maneira que ficam impossíveis de serem lida por alguém mal intencionado;
- **Autorização:** Mecanismo de filtragem, onde somente usuário com acesso ao ambiente pode acessar, é o caso do uso de ID e senha;

Os testes de segurança devem ser projetados para verificar a fundo alguma brecha na segurança de alguns desses itens, normalmente o teste de segurança é terceirizado, ficando a cargo de pessoas experientes que conheçam cada elemento de segurança de uma ampla gama de tecnologias existente.

2.3.9 Teste de Desempenho

Um usuário que acessa duas páginas sendo que uma carrega rapidamente e a outra demora consideravelmente, provavelmente este usuário irá dar mais credibilidade para a página que carregou rápido. Nada é mais frustrante do que acessar uma página que esteja fora do ar, esteja ocupado ou que acessa rapidamente e após uma requisição fique demorando para entrar em outra página. Essas situações ocorrem diariamente na *Web* e estão diretamente relacionada ao desempenho.

O teste de desempenho visa descobrir problemas que possam afetar o acesso a requisição do usuário que podem estar relacionado a falta de mecanismos do lado do servidor, largura de banda inadequada, banco de dados inadequados, sistemas operacionais, funcionalidades da *WebApp* mal projetadas e problemas de hardware ou software.

Os testes são projetados para simular situações de carga no mundo real, a medida que cresce o número de usuários simultâneos na *WebApp*, aumenta o número de downloads e uploads, o teste ajuda a responder as seguintes questões:

O tempo de resposta do servidor degrada a ponto de percepção?

Em que ponto o desempenho chega a ser inaceitável?

Quais componentes são responsáveis pela degradação de desempenho?

Qual tempo médio de resposta para usuários sob uma variedade de condições de carga?

A degradação do desempenho tem um impacto sobre a segurança do sistema?

A confiabilidade ou precisão da *WebApp* é afetada quando a carga aumenta?

O que acontece quando aplicado cargas maiores do que a capacidade máxima?

A degradação do desempenho tem efeitos no lucro da empresa?

Para descobrir as respostas a essas perguntas são feitos dois diferentes testes de desempenho: Primeiro o teste de carga que examina as cargas reais em uma variedade de níveis em uma variedade de combinações e segundo o teste de stress (força), que aumenta a carga até o ponto de ruptura para determinar com que capacidade o ambiente *WebApp* pode lidar (PRESSMAN, 2011).

2.3.10 Teste de carga

É feito um teste simples para determinar como a *WebApp* e seu ambiente do lado do servidor responderá a várias condições de carga, a seguir é definido as variáveis:

N – número de usuários

T – número de transações por usuários por unidade de tempo

D – carga de dados processado pelo servidor

Existe casos em que as variáveis são projetados para avaliar alguns fatores como: resposta média do usuário, tempo médio para download, tempo para processar uma transação. Deve-se observar essas medidas e verificar se não há alteração por uma combinação de N,T e D. Pode-se medir o resultado geral ,P, calculando da seguinte maneira

$$P = N \times T \times D$$

Para exemplificar, um site de notícias policial, em um determinado momento 10 mil usuários acessam simultaneamente uma transação T, a cada 2 minutos, a cada transação um novo download é feito, que em média tem 4kbytes, o resultado fica

$$P = [10.000 \times 0,5 \times 4] / 60 = 333,3 \text{ kbytes/segundo}$$

Portanto uma conexão que prevê esse número de usuários deve suportar esse resultado no lado do servidor (PRESSMAN, 2011).

2.3.11 Teste de esforço

O teste de esforço é uma continuação do teste de carga, mas nesse caso as variáveis N,T e D devem exceder o limite operacional, segundo Pressman (2011) a finalidade desse teste é prever se algumas situações ocorrem como:

- O sistema degrada suavemente ou o servidor desliga pela alta carga;
- O software servidor gera mensagens “servidor não disponível” ou servidor desliga quando excedido a capacidade;
- Se quando o software do servidor gera a mensagem “servidor não disponível” os usuários ficam cientes que não tem acesso ao servidor;
- O servidor coloca requisições na fila quando excede ou simplesmente não há essa opção;
- Se são perdidas transações quando a capacidade é excedida;
- Se a integridade dos dados é perdida quando a capacidade é excedida;
- Quais valores de N, T e D forçam o ambiente a falhar? Existe algo a ser melhorado em alguns desses pontos?;
- Se falhar qual é o tempo para que o sistema volte ao ar;

O teste de esforço visa verificar todas essas observações, no regime de teste é elevada a capacidade, depois diminuída rapidamente para as condições normais, em seguida elevadas novamente, ao devolver a carga ao sistema, pode-se determinar como o servidor se comporta para reunir os recursos para atender a demanda muito alta e então liberá-los quando as condições normais se restabelecerem de forma que fiquem pronto para o próximo pico (PRESSMAN, 2011).

2.4 FERRAMENTAS DE TESTES PARA ENGENHARIA WEB

Em um cenário onde há o desenvolvimento de uma aplicação sendo ela complexa ou não, a utilização de ferramentas de testes ajudam a detectar alguns erros que podem ser descoberto tardiamente, neste capítulo são abordados algumas ferramentas e exemplos de códigos para testes em uma aplicação *Web*. Os testes automatizados são programas ou *scripts* que praticam as funcionalidades da aplicação e fazem verificações automatizadas em busca de qualquer erro que apresente, os testes podem ser facilmente e rapidamente repetidos em

qualquer momento. Esta simulação permite repetir o mesmo processo quantas vezes o profissional de teste achar necessário repetir, garantindo que a aplicação não traga um comportamento indesejado. Algumas situações de testes são indispensáveis, como teste de inúmeros usuários acessando o sistema simultaneamente e carga de dados no banco de dados (BERNANDO E KON, 2008).

2.4.1 Testes automatizados com JUnit

JUnit é um *framework* open-source, criado por *Eric Gamma* e *Kent Beck* para criação de testes automatizados para linguagem de programação Java. Sua facilidade está na criação de código para automação de testes unitários com apresentação de resultados, sendo possível verificar se cada método de uma classe funciona de forma esperada, exibindo possíveis erros ou falhas.

Basicamente os testes devem ser iniciados com os testes de unidades, que é a menor parte de um código fonte, podendo ser uma função, um módulo ou classe em sistemas orientados a objeto. Este teste é muito importante para se obter uma aplicação de qualidade, pois com esses testes pode-se verificar se algum trecho de código específico realmente funcione devidamente. Na Figura 9 está ilustrado um trecho de um teste de código, usando a linguagem Java juntamente com a linguagem *JUnit*, no exemplo é usado a anotação *@test* antes de cada método.

```
public class ExemploJUnitTest {
    @Test // Este método é um caso de teste
    public void testaAdicaoDeDiasEmUmaData() throws Exception {
        SimpleDateFormat formatador = new SimpleDateFormat("dd/MM/yyyy");
        Date dataDeReferencia = formatador.parse("05/06/2008");
        Date dataDaqui5Dias = formatador.parse("10/06/2008");
        Date dataObtida = DateUtil.adicionaDiasEmUmaData(dataDeReferencia, 5);
        assertEquals(dataDaqui5Dias, dataObtida);
    }
}
```

Figura 9: Código teste

Fonte: BERNARDO E KON (2008)

Neste trecho (figura 9), é apresentado um simples código de teste de data, contendo métodos para comparar os resultados esperados, o *assertEquals* compara a igualdade do conteúdo do objeto, o *assertSame* compara se as referências se referem ao mesmo objeto, o *assertNull* verifica se a data é nula. Uma forma que facilita os testes é o uso de ferramentas integradas ao ambiente de programação. Na Figura 10 observa-se a integração do ambiente de programação Eclipse usando a ferramenta de teste *JUnit*.

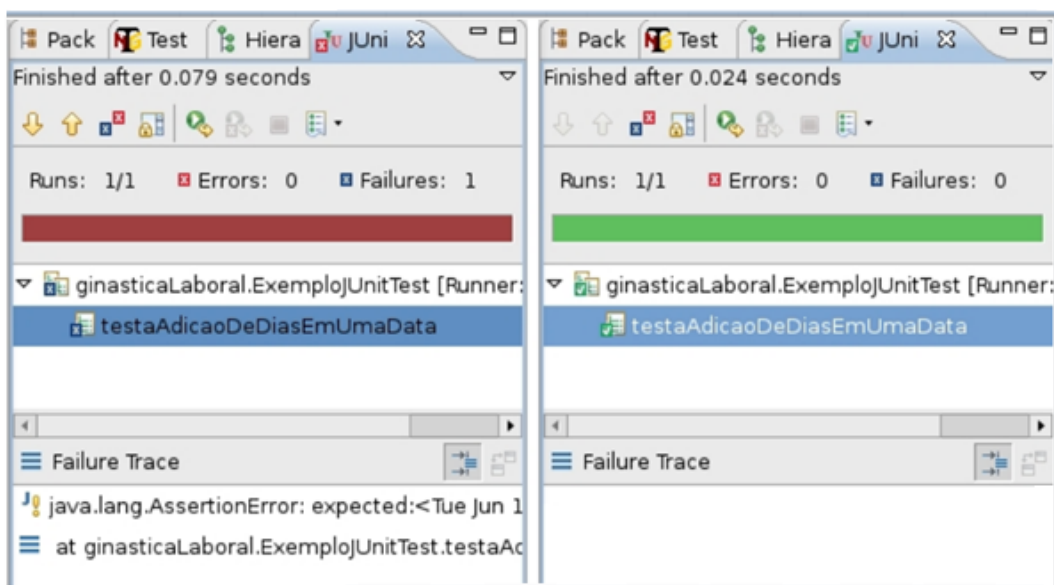


Figura 10: Relatório de testes com Eclipse e JUnit
 Fonte: BERNARDO E KON (2008)

2.4.2 Selenium RC

Selenium é uma ferramenta para testar aplicações web pelo *browser* de forma automatizada, é usado para rodar testes em um sistema finalizado. Os testes rodam diretamente no *browser*, exatamente como o usuário faria. Esta ferramenta é capaz de testar a aplicação em nível de aceitação, testa a interface do usuário, incluindo funcionalidades como um clique do mouse, digitação de teclas, uma opção selecionada, entre outras ações. Na Figura 11, é apresentado um trecho de código utilizando o Eclipse e o *Selenium RC - Selenium Remote Control e JUnit*. O código testa uma página no qual é definido um servidor local e o navegador usado, após isso é feito uma simples tarefa de acessar a aplicação, verificar se o texto está visível, clicar em um link, verificar o outro texto e fechar o navegador.

```
@Test
public void testaPaginaDoProjetoQualipso() throws Exception {
    String url = "http://www.qualipso.org";
    String servidor = "localhost";
    String porta = 4444;
    String navegador = "*firefox";
    Selenium selenium = new DefaultSelenium(servidor, porta, navegador, url);

    selenium.start(); // Abre o navegador

    // Entra no site
    selenium.open("/");
    // Verifica se um texto apareceu
    assertTrue(selenium.isTextPresent("It is all about trust"));
    // Clica em um link
    selenium.click("link=Work Areas");
    // Espera carregar a nova página
    selenium.waitForPageToLoad("30000");
    // Verifica se apareceu um outro texto
    assertTrue(selenium.isTextPresent("TRUSTWORTHY RESULTS"));

    selenium.stop(); // Fecha o navegador
}
```

Figura 11: Trecho de código usando Selenium RC e JUnit
Fonte: BERNARDO E KON (2008)

A classe *DefaultSelenium* possui métodos que abstraem o funcionamento interno de eventos dos navegadores, tais como *click* ou *select* que simulam um clique do mouse em um objeto e a seleção de uma opção em caixas de seleção, respectivamente.

2.4.3 Selenium IDE

Outro componente de testes do Selenium é o Selenium IDE, que é uma extensão do navegador Firefox. Funciona como um recorder e grava as ações do usuário na *WebApp* a ser testada. As ações podem ser transformadas em códigos em várias linguagens, inclusive em Java.

Na Figura 12 é representado o Selenium IDE. Essa ferramenta gera códigos feitos com Javascript capaz de manipular qualquer elemento como preenchimento de valores até itens com valores específicos (MARINS, 2009).

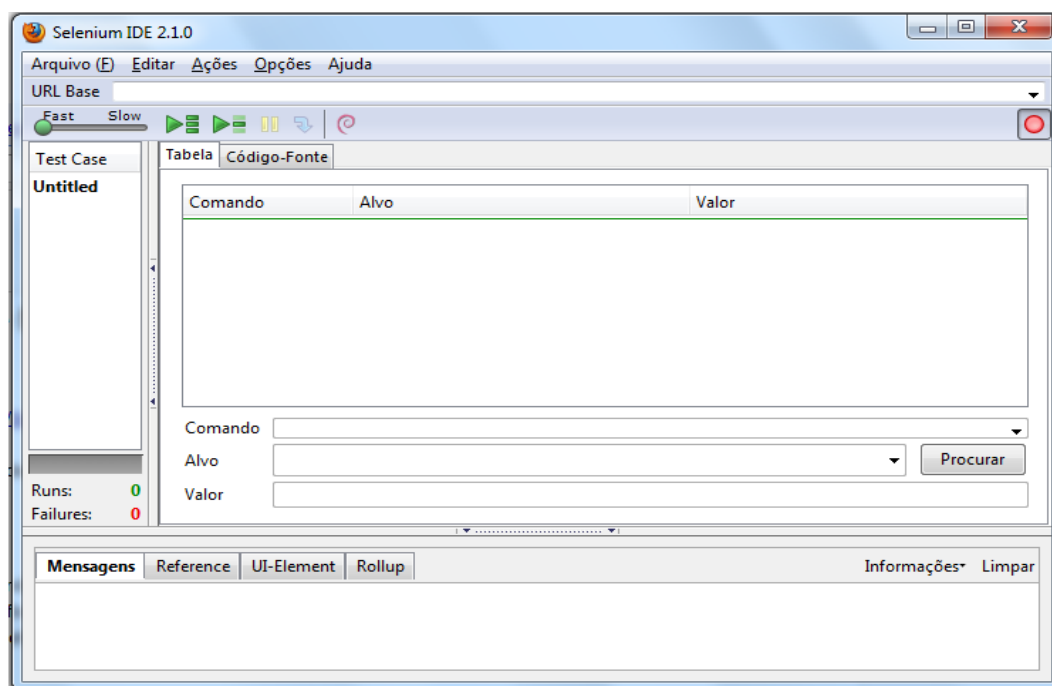


Figura 12: Selenium IDE

É adicionado a *WebApp* que será testado e realizado vários processos de execução na *WebApp* a fim de encontrar alguma irregularidade tais como preenchimento de algum campo, testar links para averiguar se estão ancorados corretamente, testar botões, mensagens de erros entre diversas outras funcionalidades em que uma *WebApp* possa disponibilizar.

É possível definir a velocidade em que os testes serão rodados além de selecionar um conjunto de vários testes de uma só vez, rodar apenas o arquivo de teste que está aberto atualmente, rodando todos os comandos do teste a partir do ponto inicial, podendo realizar um *pause* ou o *debug*. Pode ser utilizado um assistente ou diretamente no código-fonte que pode ser padrão HTML ou modificado para *Java*, *C#*, *Perl*, *PHP*, *Python* e *Ruby*.

Dentro da área branca maior é o local onde ficam os comandos na ordem em que são executados, ou seja, os comandos criados automaticamente no momento em que o botão *rec* está em execução. Pode ser inserido algum comando entre dois já existentes. Os três próximos campos são para teste, o primeiro é o item “comando”, que é a ação que é feita, como verificar algum item, abrir uma URL, entrar em algum *link* existente, entre outros comandos. O segundo campo é o “alvo” que é um valor a ser passado como parâmetro para o comando, há um botão procurar, que é muito útil quando se está desenvolvendo testes, por exemplo, verificar se uma busca por um elemento XPATH está correta e a validação esta sendo feita corretamente. O terceiro campo de entrada é o “valor” e serve para quando uma ação tem

mais de um parâmetro, basicamente serve para setar algum valor em um campo, por exemplo um *input* (MARINS, 2009).

2.4.4 Apache Jmeter

O Apache Jmeter é uma aplicação desktop *open source*, feito totalmente em Java. Originalmente foi projetada para testar aplicações *WebE* mas expandiu para outras funções de teste. Pode ser usado para testar o desempenho tanto de recursos estáticos como dinâmicos (arquivos, *Servlets*, *script*, objetos Java, base de dados e consulta, servidores FTP entre outros. Também é possível simular uma carga pesada em um servidor, para verificar se a aplicação suporta um alto tráfego de usuários. A figura 13 representa a aplicação *desktop* que será utilizada no capítulo de teste.

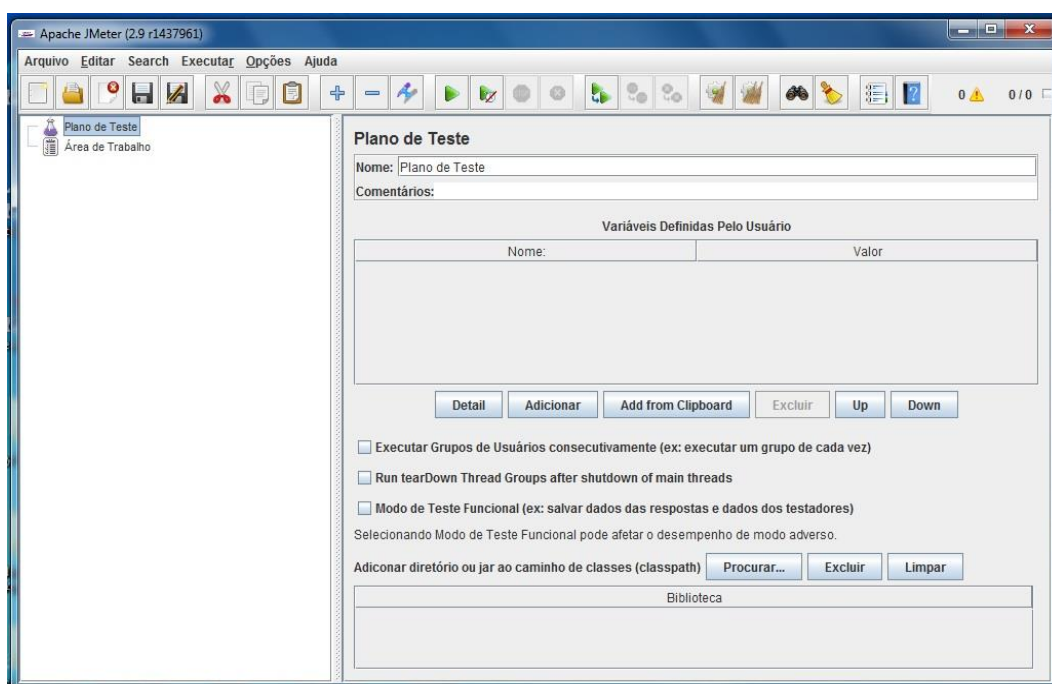


Figura 13: Apache Jmeter

2.4.5 TestComplete

É uma ferramenta completa de gerenciamento e criação de testes que possibilita a escrita de testes funcionais, testes automatizados, testes unitários entre outros. Com ela é possível criar uma suíte de teste, que pode ser composta por um ou vários projetos de testes, neste caso, se for introduzido vários projetos de teste em uma mesma suíte, podem apresentar

funcionalidades similares a serem testadas e um deles pode possuir casos de testes prontos que podem ser reaproveitados em outro projeto. Os projetos de testes comportam os *scripts* de teste, casos de teste entre outros.

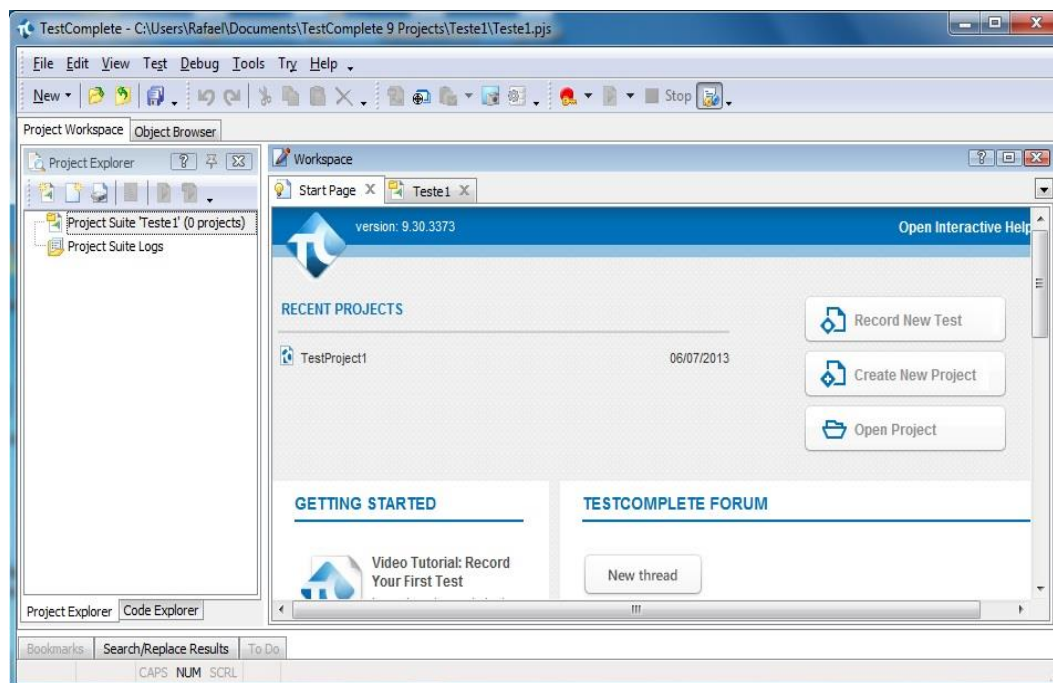


Figura 14: TestComplete

2.4.6 SikuliApi

Esta ferramenta permite testes funcionais em uma aplicação, ela permite que o testador implemente e execute casos de testes em sistema *web* e *desktop* com construção de *scripts* que serão executados automaticamente, esses *scripts* devem seguir uma ordem e ter uma finalização com o resultado esperado obtido ou alguma mensagem de erro. Esta ferramenta usa o reconhecimento de imagem para identificar e controlar os componentes. Para um melhor entendimento, a imagem 15 é exibido um pequeno *script* com uma tarefa simples.



Figura 15: Testando Sikuli

Estes comandos abre o *notepad* digita frase “Testando o Sikuli” e salva com o nome de “sikulo_script”, cada comando *click* possui uma imagem capturada da ferramenta *notepad*. Pode-se usar um script para uma simples conferencia de *link* ou até uma verificação de todo o conteúdo de uma aplicação *Web*.

2.4.7 Neoload

Neoload é uma ferramenta paga para teste de carga em aplicações *Web* que simula as atividades do usuário, possibilita a ser implantação do testes nas ferramentas como *Flex*, *Silverlight*, *GWT* e *Ajax Push*. A figura 16 apresenta tela principal da ferramenta.

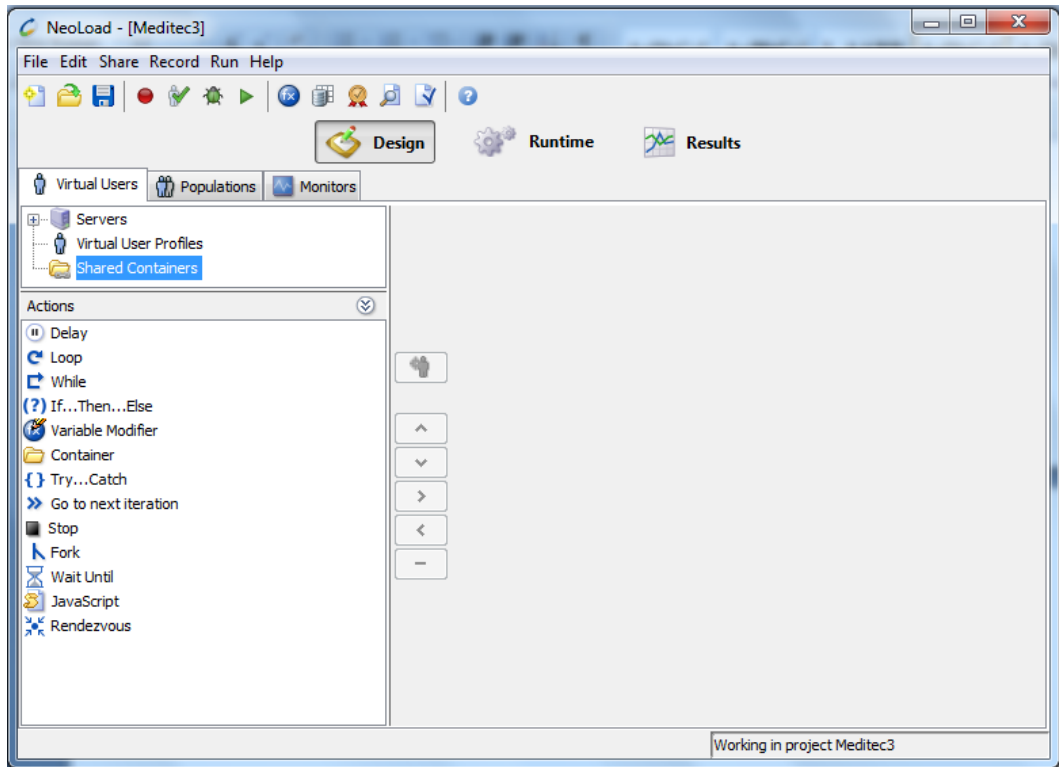


Figura 16: Teste com Neoload

3 MATERIAIS E MÉTODOS

3.1 MATERIAL

Para a realização dos testes foi utilizado um *Notebook* da marca Dell configurado com processador *Core2Duo*, com velocidade de 2.0 Gigahertz e 3 Gigabytes de memória Ram, com Sistema Operacional Windows 7 *Ultimate*. A internet utilizada continha velocidade de 10mb/s conectado através de um cabo cat5 com conector RJ 45.

O ambiente de teste utilizado foi o site do Meditec, no qual foram elaborados testes em caixa preta que são testes que não envolvem código, somente a entrada e saída de dados. Os testes utilizados foram de conteúdo, carga, e compatibilidade. Também foi elaborado um questionário para avaliação para o teste de interface.

3.2 MÉTRICA

As métricas servem para medir qualidade do que está sendo desenvolvido, que pode ser medido durante os ciclos de desenvolvimento.(CORREA, 2008)

3.2.1 Métricas de Boehm, Brown e Lipow

Boehm, Brown e Lipow apud Bueno e Campelo (1999) definem uma árvore de atributos de qualidade de software onde as direções das setas indicam implicações lógicas, por exemplo, uma aplicação que é fácil de ser entendida provavelmente será facilmente testada, entendida e modificada. A Figura 17 mostra esses atributos, contendo a portabilidade, confiabilidade, eficiência, engenharia humana, facilidade de teste, uso e modificação.

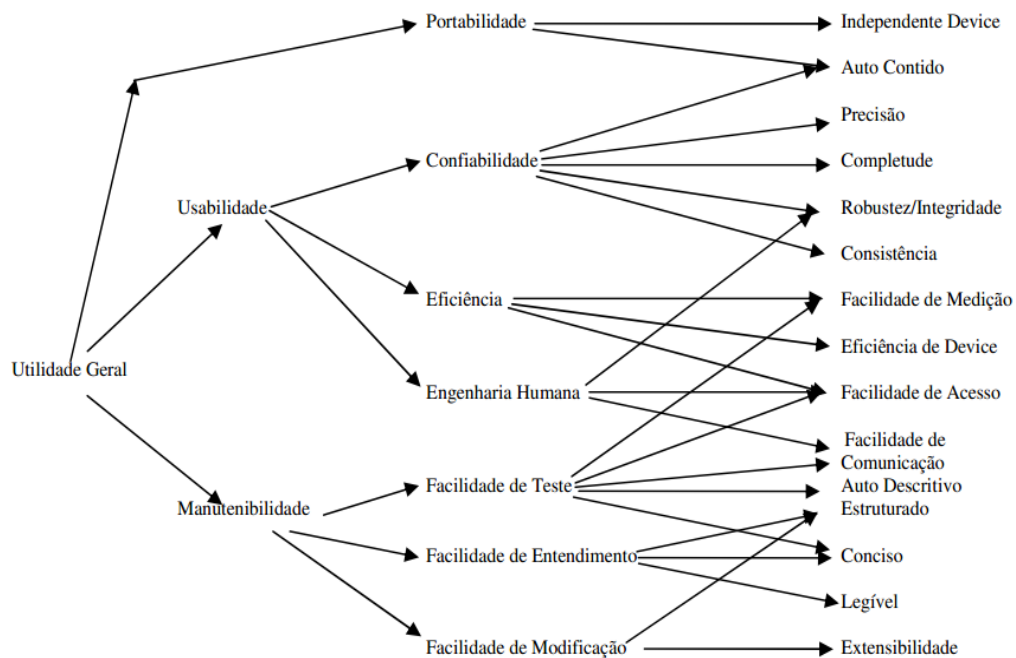


Figura 17: Árvore de atributos
Fonte: Bueno e Campelo (1999)

A estrutura de mais alto nível reflete o uso de avaliação da qualidade de software. Boehm, Brown e Lipow (1999), enfatizam a aquisição do pacote de software, o qual deve ter as seguintes características de nível médio na estrutura hierárquica: portabilidade, confiabilidade, eficiência, engenharia humana e facilidades de teste, uso e modificação.

As características de mais baixo nível são primitivas que podem ser combinadas para formar características de nível médio. As características primitivas são recomendadas como métricas quantitativas delas próprias e das características de mais alto nível (BUENO e CAMPELO apud SHNEIDERMAN, 1980)

Uma característica primitiva pode ser definida ou medida através de uma *checklist*. Valores numéricos podem ser dados aos atributos de qualidade, em alguns casos isso pode ser apropriado, como por exemplo, nos atributos de performance, e em outros pode existir um certo grau de subjetividade. Tais *checklists* podem ser úteis, mas tendem a crescer e se tornarem incômodas, surgindo a necessidade de uma organização específica e tornando-se específica para uma linguagem/sistema.

3.3 MÉTODOS

Para medir a qualidade da interface de usuário da *WebApp* foram verificadas todas as validações e máscaras da página de cadastro. Também foi verificado todo o material escrito e encontrados alguns erros que são descritos no item 3.5.1 e realizado testes usando as ferramentas (Selenium IDE e TestComplete). Para teste de usabilidade, além da compatibilidade e velocidade, foi elaborado um questionário e enviado para todos os acadêmicos ativos do curso de Tecnologia em Análise e Desenvolvimento de Sistemas e Ciências da Computação da UTFPR, público principal que utilizou a *WebApp*. A aplicação foi testada nos principais navegadores utilizados verificando a sua compatibilidade. Os aplicativos (*Jmeter e Neoload*) foram utilizados para o teste de carga.

4 RESULTADOS E DISCUSSÕES

O ambiente que foi usado para os testes foi a aplicação *Web* desenvolvida para o evento Meditec - *Medianeira in Technology* que ocorreu pela sua 4^o edição no câmpus da UTFPR em Medianeira/PR (<http://www.meditec.net.br>). A figura 18 apresenta página inicial do site.



Figura 18: Página inicial do site

4.1 TESTES

Neste capítulo apresenta o estudo experimental utilizando alguns testes realizado na aplicação *web*.

4.1.1 Teste de conteúdo

No teste de conteúdo realizado na *WebApp* foi verificado todo o conteúdo escrito do. Por ser informações locais, ou seja, as informações contidas são da própria universidade e sobre os próprios eventos que ocorreram anteriormente na universidade não houve nenhum indicio de violação autoral e nem de informações duvidosas ou confusas, também não houve conteúdo ofensivo e confuso. Todos os links foram testados e funcionam corretamente.

No teste de conteúdo realizado na *WebApp* foi verificado a ortografia dos textos contidos nas páginas, foram encontrados alguns erros ortográficos.

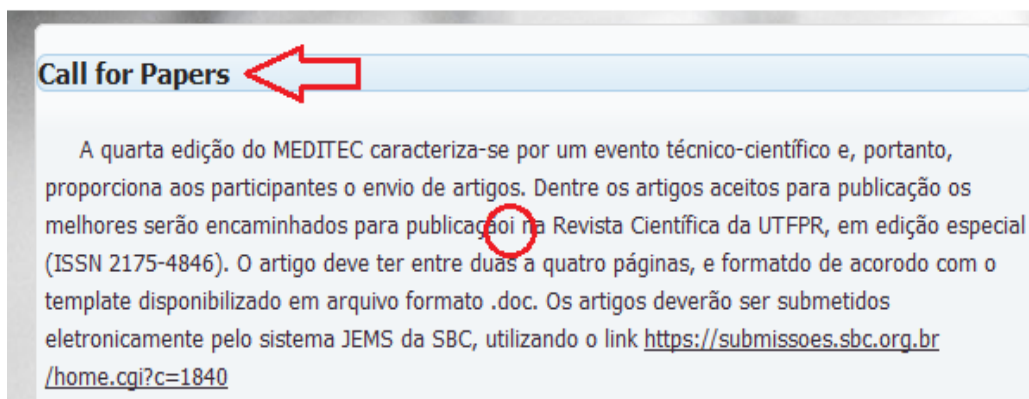


Figura 19: Erro na ortografia

Erro no título, ao invés de “Envio de Artigos ou Chamada de Artigos”, o título apresenta “Call for Papers”, como se trata de um evento regional e todo em português deveria ser mantido o padrão da língua. Também foi verificado erro de ortográfico na palavra “publicaçoi” ao invés da palavra publicação, é um erro comum na digitação, visto que algumas vezes uma letra pode ser digitada involuntariamente. Nas outras páginas da aplicação não foram encontrados erros. O texto contido nas páginas não continha alinhamento, observado na Figura 20. Esta questão é muito discutida com autores, Nielson defende que o alinhamento à esquerda é a melhor forma de padronizar uma aplicação, outros defendem o alinhamento justificado.

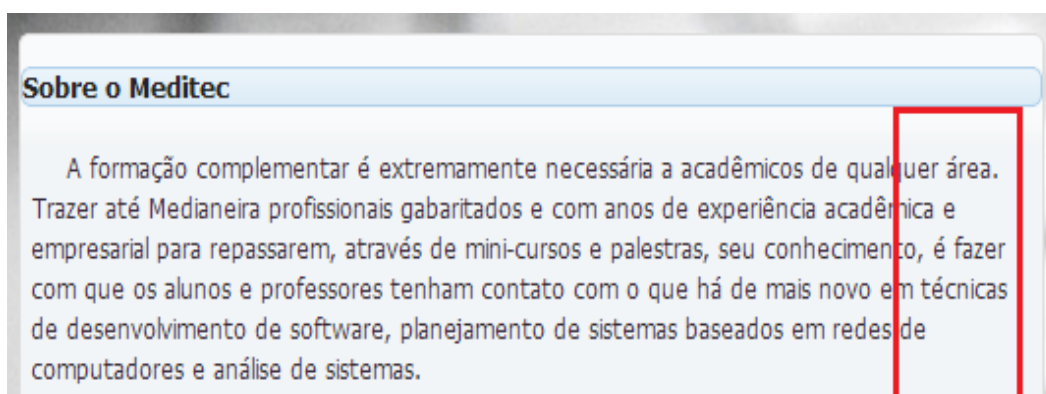


Figura 20: Alinhamento dos textos

4.1.2 Teste de interface de Usuário

A interface geral do site foi baseado em uma estilização simples, na Figura 21 é possível verificar o *html* sem estilização, como o objetivo da aplicação é informar ao usuário dados de um evento e sua utilização foi de curto prazo, os resultados foram satisfatórios.



Figura 21: Recorte do site

Foram testados todas as validações na página de cadastro, tais como CPF, senhas, telefone, e-mail, *login* e campos vazios. Também foi realizado mais de um cadastro com o mesmo CPF, o sistema não informa que o CPF já foi cadastrado, as figuras 22 exibe as mensagens de alerta dos campos que são preenchidos erroneamente.

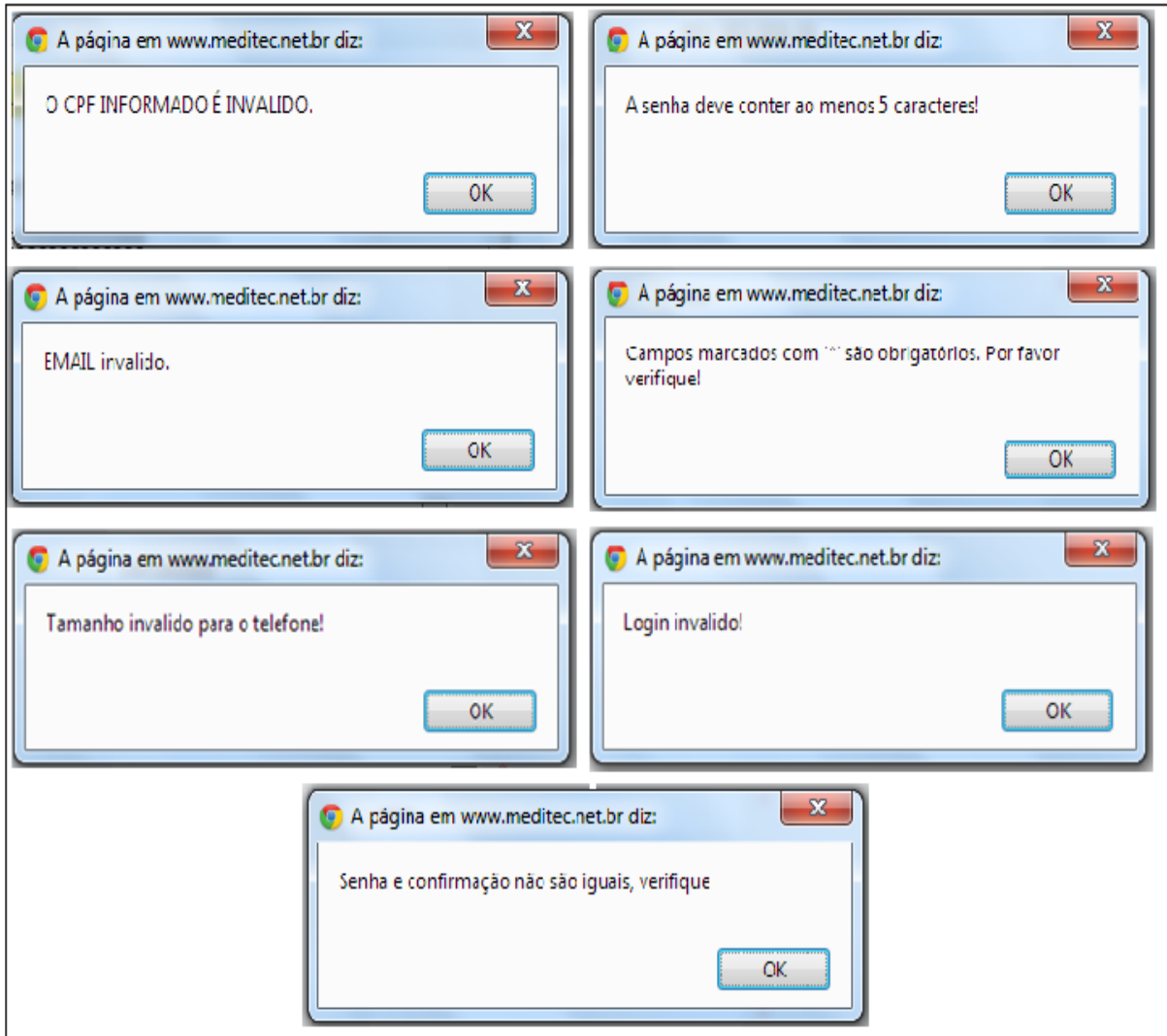


Figura 22: Teste com Selenium IDE

No teste do formulário, também foi utilizado a ferramenta *Selenium IDE* 2.2 para o preenchimento automático e verificação dos valores. A figura 23 exibe o preenchimento dos campos, o ID e o valor que é inserido automaticamente. Esta ferramenta pode ser usada para o preenchimento de formulários com vários campos onde é necessário ser testado várias vezes, assim automatizando todo o processo.

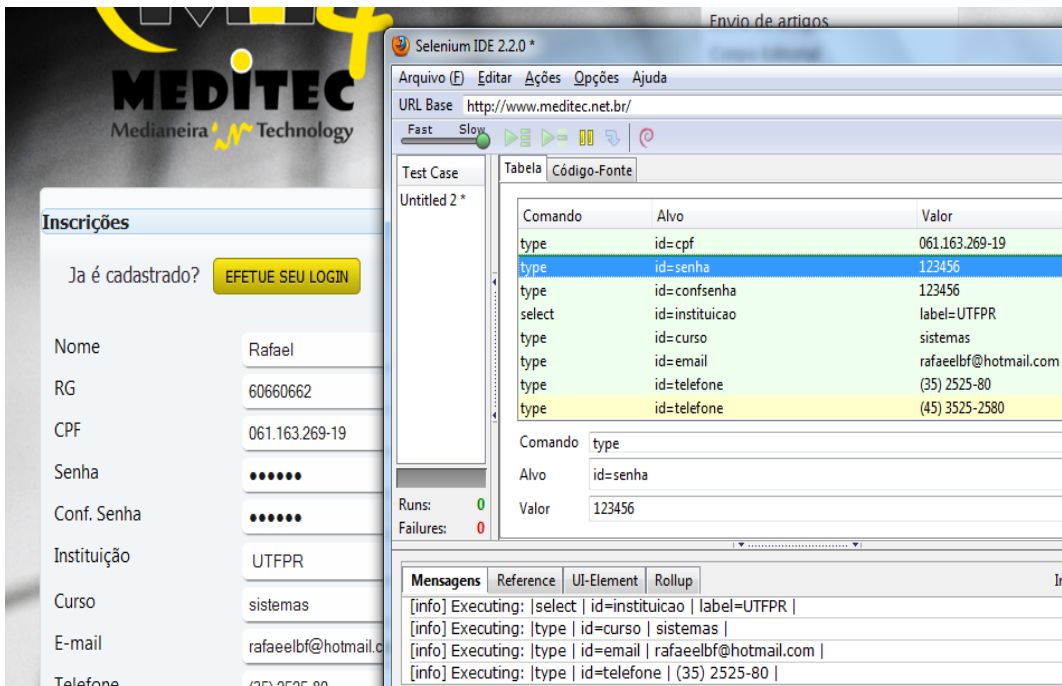


Figura 23: Teste com Selenium IDE

Outra ferramenta utilizada para este tipo de teste foi o *TestComplete*, com ele é possível gravar um caso de teste, assim como no *Selenium IDE*. A vantagem é de ser um software independente e não um *plugin* de um navegador específico, podendo assim elaborar um caso de uso em qualquer navegador, analisando as diferenças. O caso de teste foi elaborado no navegador *Google Chrome* em todas as páginas da aplicação e efetuado *login*.

Item	Operation	Value	Description
pageHttpWwwMeditecNetBr			
link	Click	...	Clicks the 'link' link.
pageHttpWwwMeditecNetBrSobreHtml	Wait	...	Waits until the browser loads the page and is ready to accept user input.
pageHttpWwwMeditecNetBrSobreHtml			
pageHttpWwwMeditecNetBrProgramaH	Wait	...	Waits until the browser loads the page and is ready to accept user input.
pageHttpWwwMeditecNetBrProgramaH			
pageHttpWwwMeditecNetBrInscricao	Wait	...	Waits until the browser loads the page and is ready to accept user input.
pageHttpWwwMeditecNetBrInscricao			
pageHttpWwwMeditecNetBrArtigosHt	Wait	...	Waits until the browser loads the page and is ready to accept user input.
pageHttpWwwMeditecNetBrArtigosHt			
pageHttpWwwMeditecNetBrCorpoHtml	Wait	...	Waits until the browser loads the page and is ready to accept user input.
pageHttpWwwMeditecNetBrCorpoHtml			
link	Click	...	Clicks the 'link' link.
pageHttpWwwMeditecNetBrPalestran	Wait	...	Waits until the browser loads the page and is ready to accept user input.
pageHttpWwwMeditecNetBrPalestran			
link	Click	...	Clicks the 'link' link.
pageHttpWwwMeditecNetBrContatoHt	Wait	...	Waits until the browser loads the page and is ready to accept user input.
pageHttpWwwMeditecNetBrContatoHt			
link	Click	...	Clicks the 'link' link.
pageHttpWwwMeditecNetBrInscricao	Wait	...	Waits until the browser loads the page and is ready to accept user input.

Figura 24: TestComplete

Na figura 24 estão listadas algumas das ações que foram montadas no caso de teste, onde o “Item” é a página que está sendo acessado, a “Operação” que no caso foi *click* do mouse, a espera da página (*Wait*) sendo carregada e “*Description*” a descrição da ação que pode ser um clique do mouse, preenchimento de algum campo, etc.



Figura 25: Ações gravadas no TestComplete

A figura 25 ilustra as ações, ou seja, captura o momento do “click” do *mouse* nos *links* da página. A cada *click* do *mouse* é capturado e apresentado na aplicação. Com esta aplicação pode ser gerado vários casos de teste, preencher vários formulários, pode ser usado durante a programação de uma aplicação, automatizando os testes sem a necessidade de ficar preenchendo manualmente os campos.

4.1.3 Teste de usabilidade

Para a avaliação do teste de usabilidade, foi elaborado um questionário utilizando o *Google Doc* (questionário online para colaboração em tempo real com outros usuários) com 11 perguntas relacionadas ao uso e desempenho da *WebApp* (anexo 1), o questionário foi enviado por email para todos os alunos ativos cadastrados no Ambiente *Moodle* da Universidade Tecnológica Federal do Paraná - campus Medianeira. Após um período de 4 dias foram contabilizados 24 respostas para extração e apresentação dos resultados. As perguntas continham uma escala de 1 a 5, classificados em “Muito Ruim”, “Ruim”, “Regular”, “Bom”, e “Muito Bom”, com campos para observação ou sugestão em cada pergunta.

Questão n° 1: *Layout*

Dentre as respostas sobre os mecanismos de navegação, conteúdo (Figura 26), 20 pessoas responderam entre 3 e 4 pontos, “Regular e Bom”, uma média de 42%, 1 pessoa respondeu que estava “Ruim”, e 3 pessoas responderam que estava “Muito Bom”.

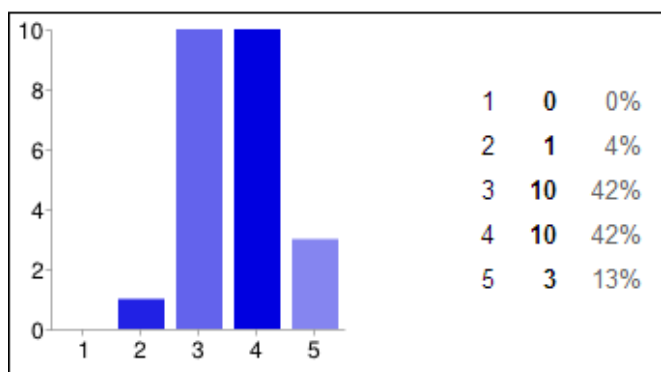


Figura 26: Gráfico de resposta 1

Questão n° 2: Clareza

Quanto a clareza, os textos presentes na *WebApp* eram de fácil entendimento para 46% “Bom”, 33% “Muito Bom”, 17% “Regular”, uma resposta classificada como “Ruim” que corresponde a 4% (Figura 27).

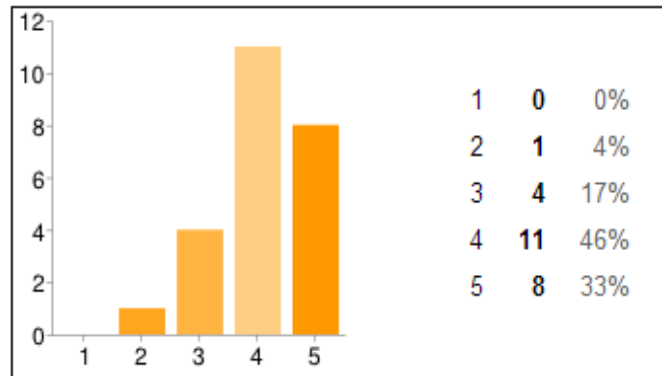


Figura 27: Gráfico de resposta 2

Questão n° 3: Ergonomia – estética

Quanto a estética, ou seja, se as cores, tipo de letra e imagens estão de acordo com o que a *WebApp* foi designada, se os usuários se sentem “confortáveis” ao navegar, 1 pessoa respondeu que está “Muito Ruim”(4%), 17% responderam que está “Ruim”, 22% responderam que está “Regular”, a maioria (48%) responderam que está “Bom” e 9% responderam que está “Muito Bom”(Figura 28). Algumas observações dos usuários:

“Pouca margem, texto não justificado, background ruim, resumindo layout está muito ruim!”

“Não é algo exagerado, só existe muitos recursos visuais misturados gerando uma poluição visual tirando assim o foco do conteúdo”

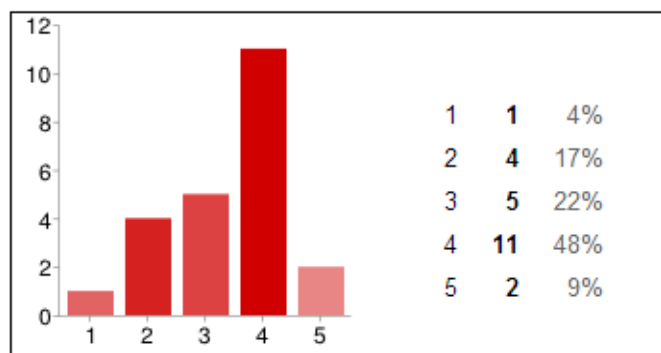


Figura 28: Gráfico de resposta 3

Questão n° 4 – Otimização

Sobre a otimização da *WebApp* em relação a resolução da tela (Figura 29), 46% classificaram como “Bom”, contra 13% “Muito Bom”, 25% “Regular”, 4% “Ruim” e 13% “Muito Ruim”, também houve uma observação de um usuário:

“Para total otimização precisaria ser um layout responsivo, entretanto o atual aproveita bem esse quesito ainda que não seja o mencionado”

Design Responsivo é uma técnica de estruturação usando HTML e CSS, consiste na adaptação do site ao *browser* em que o usuário está acessando sem que seja necessário definir várias folhas de estilos específica para cada resolução. É um tipo de design em que o *layout* fica variável de acordo com a resolução do usuário (ARRIGONI, 2012).

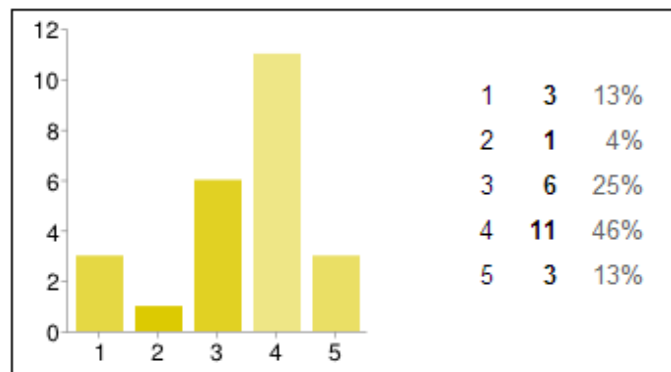


Figura 29: Gráfico de resposta 4

Questão n° 5: Desempenho

Em relação ao desempenho da *WebApp* se o tempo de resposta foi aceitável. 42% como classificaram “Bom” e 33% “Muito Bom”, apenas 1 usuário (4%) marcou “Muito Ruim” e 21% “Regular” (Figura 30).

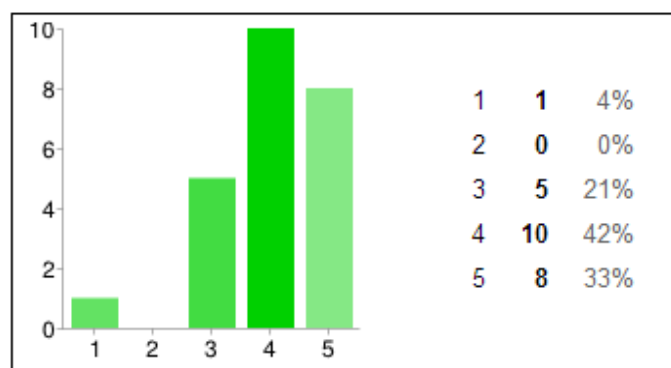


Figura 30: Gráfico de resposta 5

Questão n° 6 - Semântica

Referente se o usuário encontrou tudo o que procurava, apenas 1 (4%) respondeu “Muito Ruim”, 13% responderam “Ruim”, 21% responderam “Bom” e 29% responderam “Muito Bom”. Não houve comentários dos usuários que responderam “Ruim” e “Muito Ruim” (Figura 31).

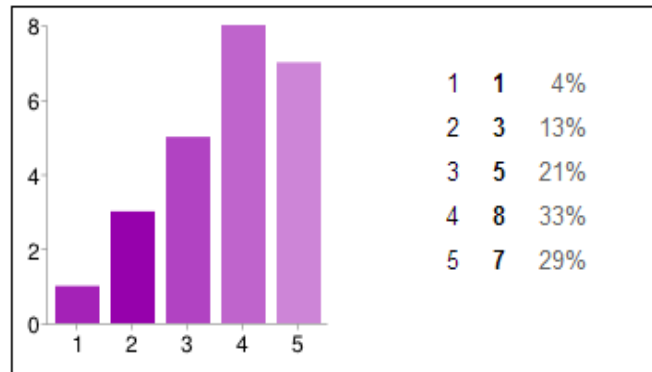


Figura 31: Gráfico de resposta 6

Questão n° 7–Estrutura

A Figura 32, verifica se o cadastro esta de acordo com a finalidade da *WebApp*. A maioria dos usuários, cerca de 58% responderam “Bom”, contra 8% “Ruim”, 13% “Regular” e 21% “Muito Bom”. Não houve comentários dos usuários

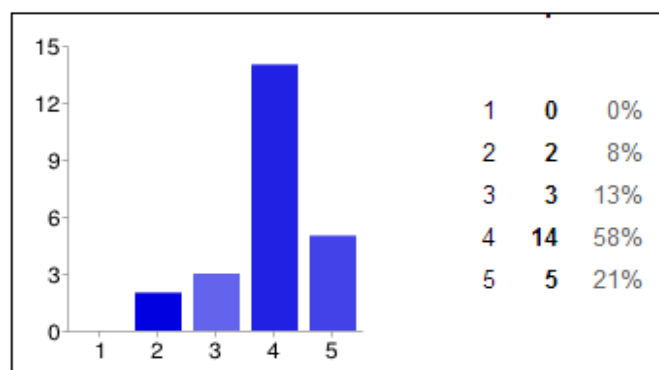


Figura 32: Gráfico de resposta 7
Fonte: Autoria Própria

Questão n° 8 – Objetivos do site

Se os objetivos do site estão claros e foram reconhecidos rapidamente, 1 usuário (4%) respondeu “Muito Ruim”, nenhum respondeu “Ruim”, 13% marcou “Regular”, 52%

respondeu “Bom” e 30% responderam “Muito Bom”. Não houve nenhuma observação dos usuários (Figura 33).

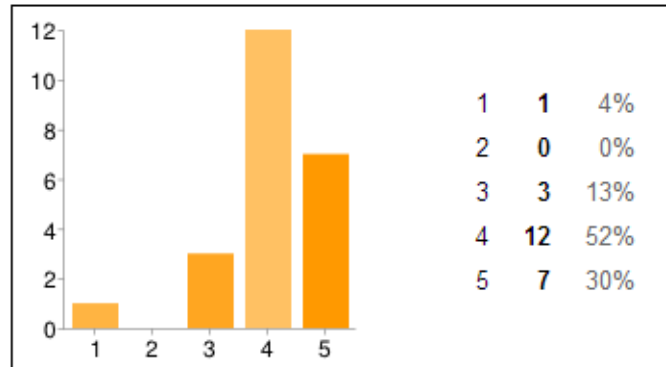


Figura 33: Gráfico de resposta 7

Questão n° 9 - Compatibilidade

A figura 34 apresenta a compatibilidade bem distribuída, 50% notaram alguma diferença e 50% não notaram nenhuma diferença ao usar navegadores diferentes.

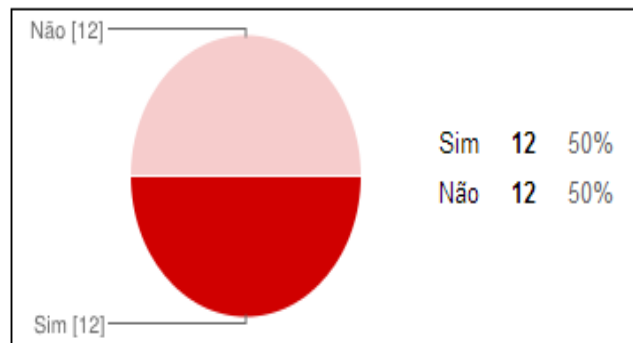


Figura 34: Gráfico de resposta 9

Houve apenas uma resposta sobre a questão.

“ Apenas cuidado em relação aos alertas do JavaScript. Eles se tornam irritantes em alguns navegadores quando demasiados (tal qual são na escolha dos cursos e palestras)”.

Questão n° 10 - links

Considerando o funcionamento correto dos *links*, 91% responderam que todos funcionaram corretamente contra 9%. Não foi informado pelos usuários que responderam “não” quais *links* não funcionaram (Figura 35).

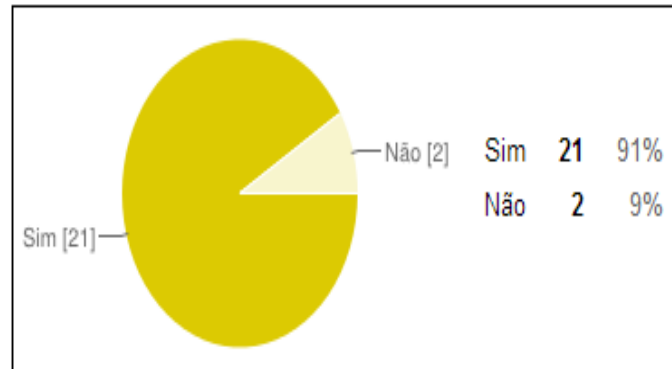


Figura 35: Gráfico de resposta 10

Questão n° 11 - Acessibilidade

Quanto ao suporte da aplicação para pessoas com necessidades especiais. A figura 36 representa 83% não souberam identificar e 17% informaram que a aplicação não tem suporte para necessidades especiais.

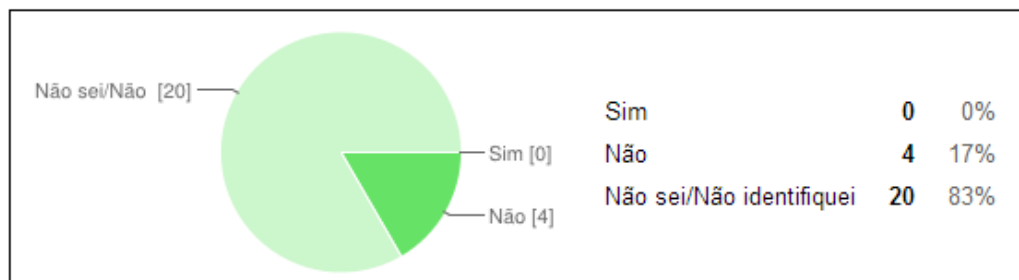


Figura 36: Gráfico de resposta 11

4.1.4 Teste de compatibilidade

Para iniciar o teste de compatibilidade, a *WebApp* foi testada nos mais conhecidos navegadores existentes na internet, entre eles o Mozilla Firefox, Google Chrome, Internet Explorer, Safari e Opera, em geral não foram apresentadas diferenças significativas, apenas em uma das páginas em que as tabelas não apresentaram o mesmo tamanho, pode-se ver essa diferença na imagem 37, no Mozilla Firefox e Internet Explorer ficou pequeno, já nos outros navegadores ficaram normais.

Mozilla Firefox

Corpo Editorial	
Nome	Afiliação
Alceu Britto Jr	Pontifícia Universidade Católica do Paraná
Claudia Rizzi	Unioeste
Edirlei Soares de Lima	PUC-Rio
Egon Hilgenstieler	Universidade Federal do Paraná

Google Chrome

Corpo Editorial	
Nome	Afiliação
Alceu Britto Jr	Pontifícia Universidade Católica do Paraná
Claudia Rizzi	Unioeste
Edirlei Soares de Lima	PUC-Rio
Egon Hilgenstieler	Universidade Federal do Paraná
Everton Coimbra de Araújo	Universidade Tecnológica Federal do Paraná

Figura 37: Compatibilidade de navegadores

Durante a navegação no Internet Explorer foi encontrado erros. A imagem 38 ilustra a página inicial da aplicação, foi verificado que várias imagens continham um contorno em azul.



Figura 38: Navegador Internet Explorer

4.1.5 Teste de carga e *stress*

Foi utilizado o *Jmeter* para fazer o teste de carga em cima da aplicação. A ferramenta possui alguns elementos básicos para realizar testes, como o *threadGroup*, que contém a configuração de um grupo de usuários virtuais. É necessário uma requisição *http*, no *Jmeter* é usado o *http Request*, que permite o envio de requisição HTTP/HTTPS ou arquivos para um servidor Web, no caso será direcionando para o link (www.meditec.com.br). Para verificar os dados, é necessário adicionar um ouvinte, esses resultados podem ser mostrados de várias formas, entre elas através de tabela, gráfico, árvore, etc. A figura 39 ilustra uma plano de teste que foi elaborado neste estudo. Para apresentar os dados foram aplicados os resultado em tabela e gráfico.

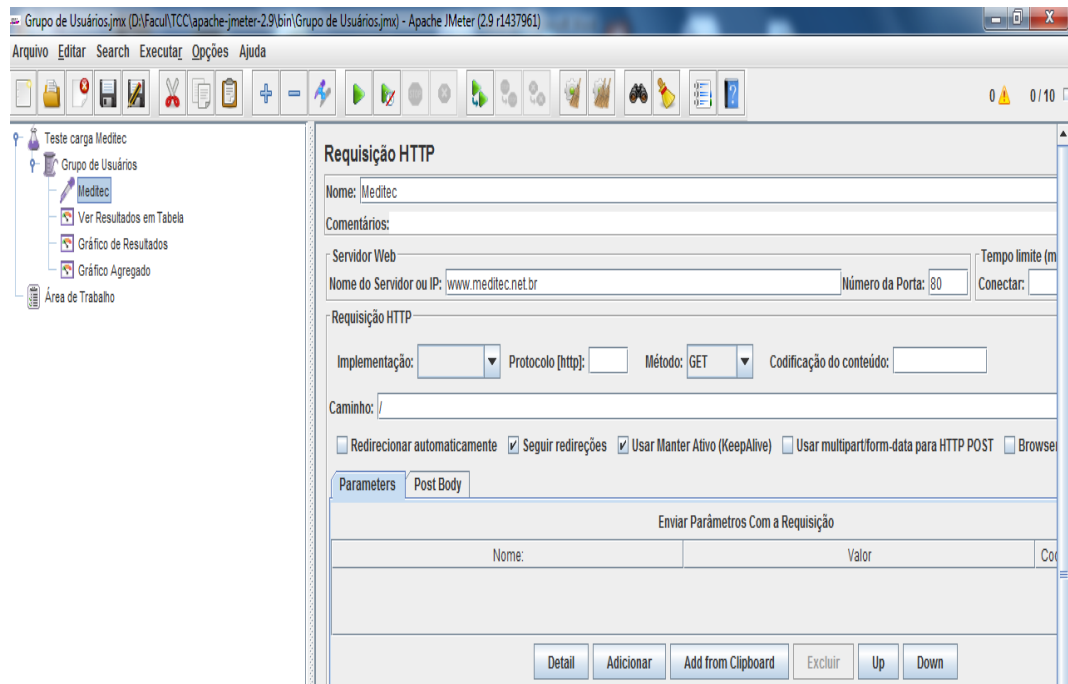


Figura 39: Teste carga com Jmeter

4.1.5.1 Teste 1 com Jmeter

Para o primeiro teste foi definido um número de 200 usuários acessados em um tempo de 100 segundos. Na figura 40 pode-se observar na coluna “estado” todos os acessos foram bem sucedidos, com 0% de erro, obteve uma média de aceso de 642 milissegundos entre os 200 acessos, o mínimo foi de 516 e o máximo de 3535.

Amostra #	Tempo de início	Nome do Usuário...	Rótulo	Tempo da amostr...	Estado	Bytes	Latency		
1	17:20:34.981	Grupo de Usuário...	Meditec	579	🟢	4126	578		
2	17:20:34.859	Grupo de Usuário...	Meditec	701	🟢	4126	700		
3	17:20:35.509	Grupo de Usuário...	Meditec	516	🟢	4126	516		
4	17:20:35.981	Grupo de Usuário...	Meditec	525	🟢	4126	524		
5	17:20:36.488	Grupo de Usuário...	Meditec	516	🟢	4126	516		
6	17:20:37.078	Grupo de Usuário...	Meditec	516	🟢	4126	516		
7	17:20:37.578	Grupo de Usuário...	Meditec	526	🟢	4126	526		
8	17:20:38.578	Grupo de Usuário...	Meditec	520	🟢	4126	520		
9	17:20:39.077	Grupo de Usuário...	Meditec	530	🟢	4126	530		
10	17:20:39.579	Grupo de Usuário...	Meditec	529	🟢	4126	529		
11	17:20:40.078	Grupo de Usuário...	Meditec	530	🟢	4126	530		
12	17:20:40.579	Grupo de Usuário...	Meditec	540	🟢	4126	540		
13	17:20:41.079	Grupo de Usuário...	Meditec	534	🟢	4126	534		
14	17:20:38.086	Grupo de Usuário...	Meditec	3535	🟢	4126	3535		
15	17:20:41.579	Grupo de Usuário...	Meditec	530	🟢	4126	530		
16	17:20:42.580	Grupo de Usuário...	Meditec	562	🟢	4126	562		
17	17:20:43.079	Grupo de Usuário...	Meditec	536	🟢	4126	536		
18	17:20:43.579	Grupo de Usuário...	Meditec	527	🟢	4126	527		
19	17:20:44.078	Grupo de Usuário...	Meditec	525	🟢	4126	525		
20	17:20:44.578	Grupo de Usuário...	Meditec	533	🟢	4126	533		
21	17:20:45.078	Grupo de Usuário...	Meditec	530	🟢	4126	530		
22	17:20:42.079	Grupo de Usuário...	Meditec	3534	🟢	4126	3534		
23	17:20:45.577	Grupo de Usuário...	Meditec	576	🟢	4126	576		
24	17:20:46.077	Grupo de Usuário...	Meditec	537	🟢	4126	537		
25	17:20:46.577	Grupo de Usuário...	Meditec	539	🟢	4126	539		
26	17:20:47.577	Grupo de Usuário...	Meditec	529	🟢	4126	528		
27	17:20:47.077	Grupo de Usuário...	Meditec	1342	🟢	4126	1342		
28	17:20:48.078	Grupo de Usuário...	Meditec	532	🟢	4126	532		
29	17:20:48.577	Grupo de Usuário...	Meditec	545	🟢	4126	545		
Rótulo	# Amostras	Média	Mediana	Linha de 90%	Mín.	Máx.	% de Erro	Vazão	KB/s
Meditec	200	642	533	651	516	3535	0,00%	2,0/sec	8,1
TOTAL	200	642	533	651	516	3535	0,00%	2,0/sec	8,1

Figura 40: Tabela de resultados 1

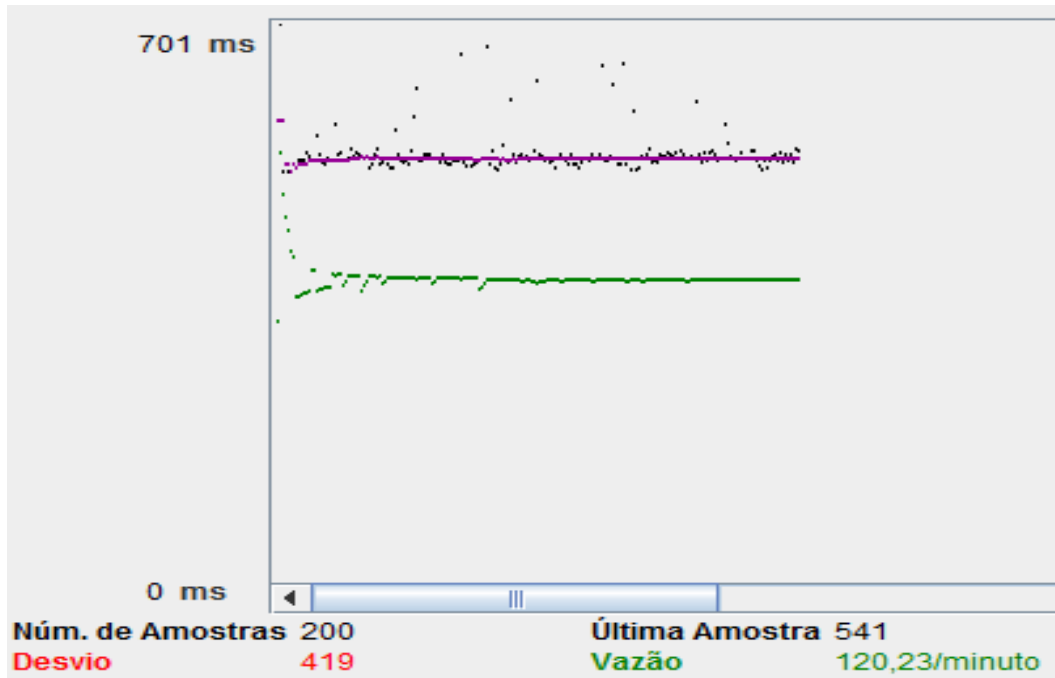


Figura 41: Gráfico de resultados 1

Na figura 41 é possível observar o gráfico, que plota a mediana que se manteve na média (em roxo), os pontos em preto são os desvios, alguns acessos mais lentos e outros mais rápidos. A vazão (em verde) também se manteve, com um acesso de 120 usuários por minuto.

4.1.5.2 Teste 2 com Jmeter

Em um segundo caso, foi elaborado um teste com mais usuários, para testar o *stress*, foi sendo incrementado gradativamente o número de usuários com o mesmo tempo (100 segundos) para todos os usuários, verificou-se que acima de 970 usuários acessando a *WebApp* começou a apresentar erros de acesso, na figura 42 é possível ver que mais de 30% dos acessos deram erro.

Rótulo	# Amostras	Média	Mediana	Linha de 90%	Mín.	Máx.	% de Erro	Vazão	KB/s
Meditec	970	1359	542	1896	514	15757	30,31%	8,6/sec	29,2
TOTAL	970	1359	542	1896	514	15757	30,31%	8,6/sec	29,2

Figura 42: Tabela de resultados 2

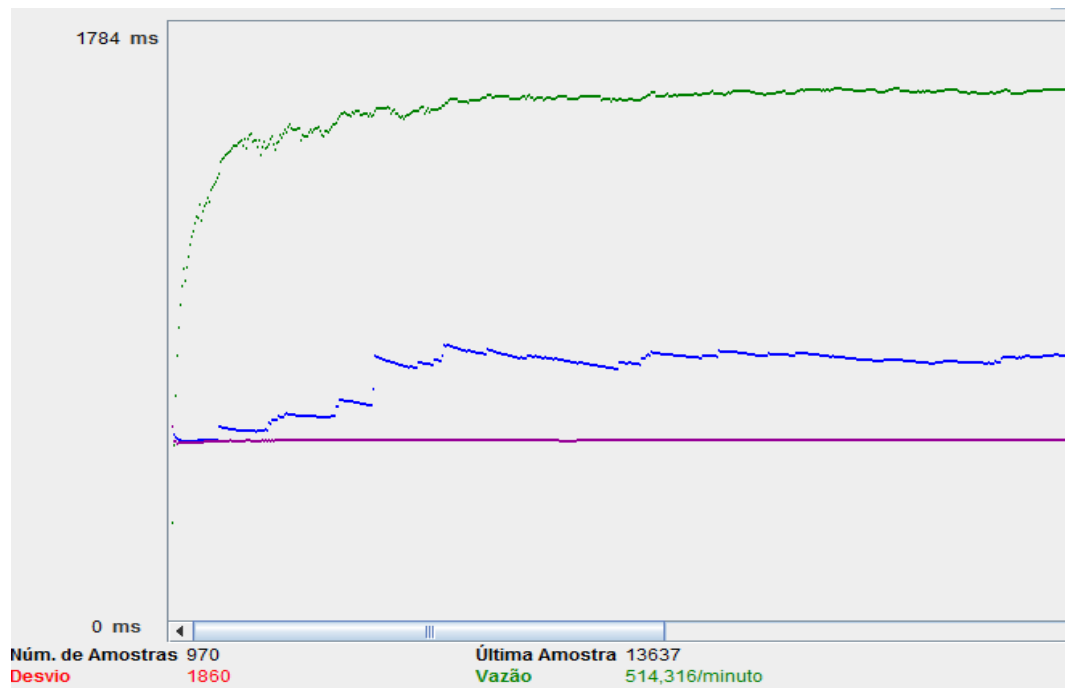


Figura 43: Gráfico de resultados 2

Foi necessária uma vazão superior a 500 usuários por minuto para forçar erros no acesso a aplicação, no gráfico (figura 43) é possível observar a vazão em verde que foi aumentando gradativamente. A média de acesso foi de 542 milissegundos.

4.1.5.3 Teste 1 com *Neoload*

Outra ferramenta utilizada nos testes foi o *Neoload*, por se tratar de uma ferramenta paga, foi utilizada a versão de teste que é limitado em 10 usuários. Foi criado três cenários de testes, um de acesso com um usuário, outro com acesso gradual iniciando com um usuário e terminando com dez usuários e por fim um cenário com dez usuários simultâneos.

Na Figura 44 é apresentado um resumo do teste com um usuário. O cenário foi montado acessando todas as páginas da aplicação e efetuado o *login*, com uma duração de 2 minutos, algumas páginas foram repetidas durante a elaboração do teste, sendo no total 14 páginas visualizadas e durante o teste foi visualizado 91 vezes entre as páginas e o *login*, obtendo uma média de 1,40mb/s com resposta de 355 milissegundos.

Total pages	14	—	Average pages/s	0,1
Total hits	91	—	Average hits/s	0,8
Total users launched	1	—	Average Request response time	0,355 s
Total iterations completed	2	—	Average Page response time	1,22 s
Total throughput	2.1 MB	—	Average throughput	0,14 Mb/s
Total hit errors	0	—	Error rate	0 %
Total action errors	0	—	Total duration alerts	0 %

Figura 44: Resumo de teste 1

Na Figura 45 é exibido o gráfico do teste ao longo de sua duração, o primeiro gráfico apresenta somente 1 usuário, e o segundo apresenta os acessos (verde) e as respostas (amarelo) onde obteve picos de resposta.

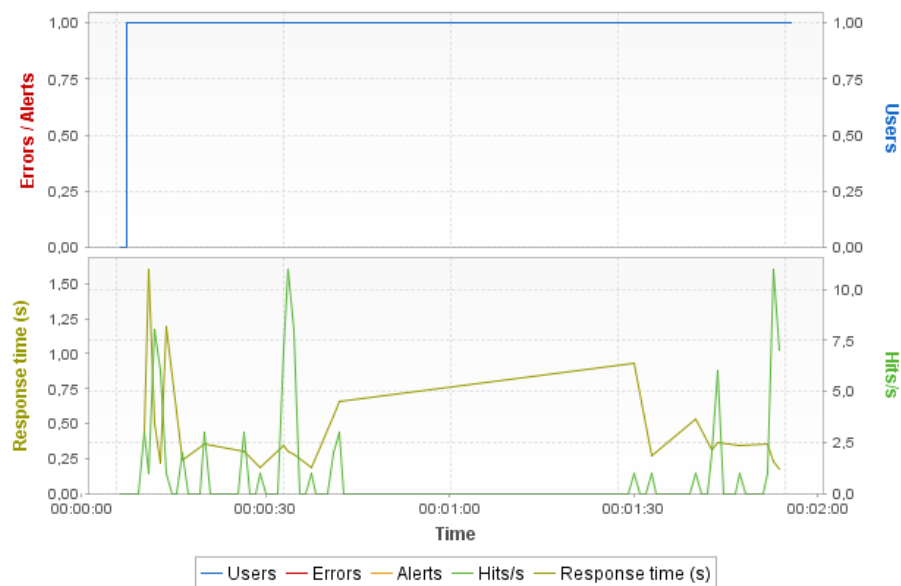


Figura 45: Gráfico teste 1
Fonte: Autoria Própria

4.1.5.4 Teste 2 com Neoload

Neste exemplo foi testado usando 10 usuários, iniciando com 1 usuário e incrementando mais 1 a cada 20 segundos, a figura 47 ilustra esse incremento através de um gráfico.

Total pages	95	—	Average pages/s	0,8
Total hits	666	—	Average hits/s	5,5
Total users launched	10	—	Average Request response time	0,393 s
Total iterations completed	12	—	Average Page response time	1,39 s
Total throughput	19.62 MB	—	Average throughput	1,30 Mb/s
Total hit errors	0	—	Error rate	0 %
Total action errors	0	—	Total duration alerts	0 %

Figura 46: Resumo teste 2

A figura 46 apresenta a média dos resultados obtidos no teste, com um total de 666 páginas acessadas durante 2 minutos, a média de resposta se manteve um pouco mais lento do que o primeiro teste, obtendo uma média de 1,30mb/s e uma resposta de 393 milissegundos.

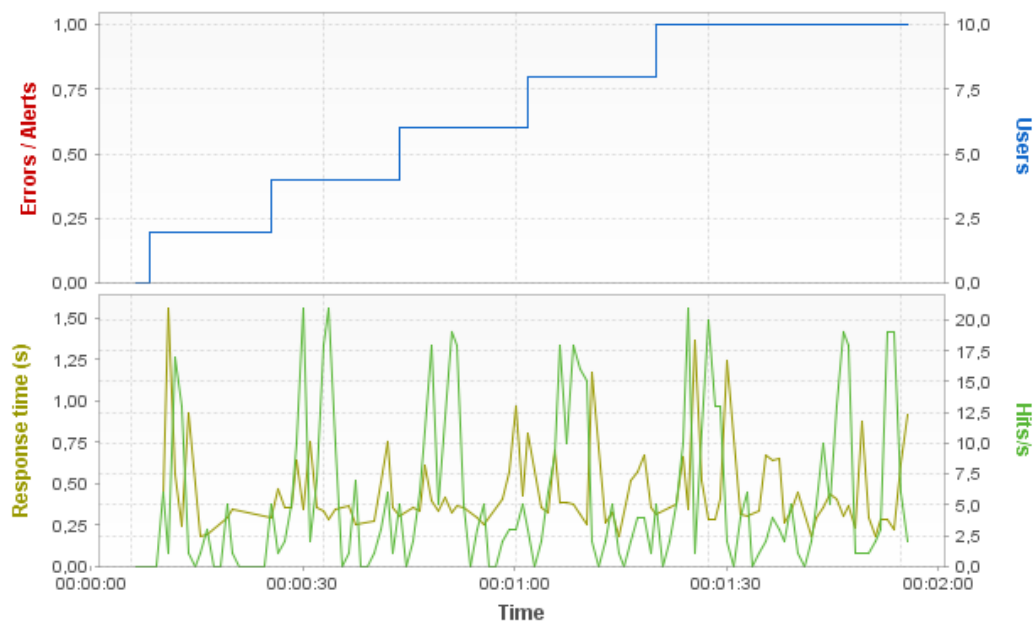


Figura 47: Gráfico teste 2

4.1.5.5 Teste 3 com Neoload

Este cenário foi executado utilizando os 10 usuários virtuais simultaneamente, o que gerou um pouco mais de carga, na figura 48 pode-se observar que a média de acesso subiu para 1,38mb/s com tempo de resposta de 576 milissegundos.

Total pages	127	—	Average pages/s	1,0
Total hits	715	—	Average hits/s	5,9
Total users launched	10	—	Average Request response time	0,576 s
Total iterations completed	16	—	Average Page response time	1,69 s
Total throughput	20.82 MB	—	Average throughput	1,38 Mb/s
Total hit errors	0	—	Error rate	0 %
Total action errors	0	—	Total duration alerts	0 %

Figura 48: Resumo teste 3

Na figura 49 pode-se observar os 10 usuários com acesso simultâneo e o gráfico de tempo de resposta na duração de 2 minutos.

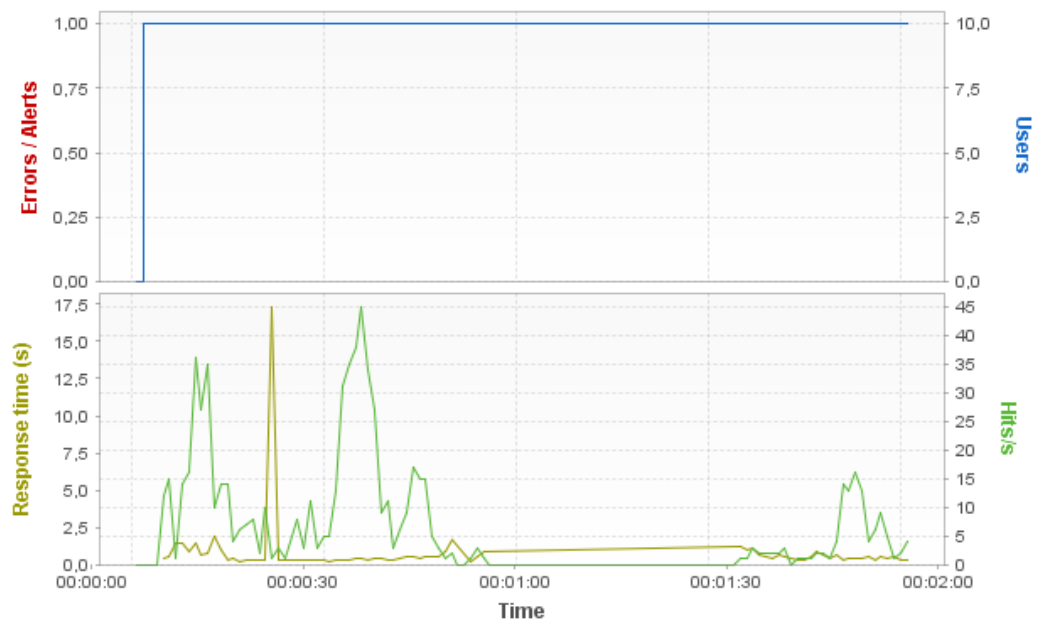


Figura 49: Gráfico Teste 3

Conclui-se que a diferença entre um até dez usuários acessando a aplicação houve uma pequena queda na velocidade de acesso. 221 milissegundos a mais que o primeiro teste e 183 a mais que o segundo teste.

Com o resultado do questionário, foi obtido uma média de 3 a 4 pontos e algumas de 4 a 5 pontos de uma escala de 1 a 5, houve alguns comentários descritos no trabalho, como o texto não justificado e pouca margem, site responsivo que se ajusta a diversas resoluções de tela, desde *smartphones* até grandes monitores. Metade dos usuários sentiram diferença ao usar navegadores diferentes, uma dessas diferenças foi descrito no trabalho.

Faltou um pouco de estilização de uma forma geral, algum estilo de letra ou fundo mais “amigável”, algumas páginas apresentaram *html* puro. Faltou um *link* para o usuário

acessar o *Backoffice* direto ou mesmo um campo de usuário e senha na página inicial, normalmente em uma aplicação *Web* existe um *link* no canto superior direito para o usuário se logar no sistema. Na aplicação o usuário tem que acessar o site, clicar no link “cadastro”, clicar no link “*login*” e por fim inserir os dados para logar.

5 CONCLUSÕES

Neste capítulo tem por finalidade apresentar a conclusão sobre testes em *WebApp* e sugestões para trabalhos futuros

5.1 CONSIDERAÇÕES FINAIS

Todo o material descrito neste trabalho e testes realizados tem a finalidade de apresentar o quanto os testes em uma aplicação *Web* são importantes e o quanto pode custar caso esse item possa ser deixado de lado, em um simples teste realizado na aplicação foi verificado erros na ortografia e validação. Alguns como texto não justificado, erro de digitação e título em inglês, como se trata de um evento regional e todo em português poderia ser mantido o padrão da língua.

Os testes de carga realizados mediram o acesso a aplicação, com os resultados obtidos conclui-se que obteve uma boa média de acesso, sendo necessário mais de 500 usuários por minuto para gerar algum tipo de erro no acesso. A média de resposta entre os testes de cargas se manteve entre 350 a 580 milissegundos.

Este trabalho demonstrou algumas formas de testes que podem ser aplicado no desenvolvimento de uma aplicação *Web* visando. Também demonstrou na forma de um questionário preenchido o quão importante é a opinião de usuários, extraindo pontos positivos e negativos para incrementar no protótipo da aplicação.

Conclui-se que, combinando métodos de testes e engenharia descritos neste trabalho, é possível criar uma aplicação *Web* de alta qualidade e confiável.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Para trabalhos futuros, pode ser desenvolvido casos de teste com outras ferramentas e elaborar testes em caixa branca que são testes a nível de código usando por exemplo o Selenium RC. Também pode estar sendo explorado testes para aplicações com *layout* responsivo, incluindo também testes para aplicativos móveis.

6 REFERÊNCIAS BIBLIOGRÁFICAS

ARRIONI, R. **Responsive Design: Dicas para tornar seu site acessível em qualquer resolução.** Disponível em: <<http://www.devmedia.com.br/responsive-design-dicas-para-tornar-seu-site-acessivel-em-qualquer-resolucao/28316>>. Acessado em 2 de agosto de 2013

BERNARDO, P.C; Kon, F. **A importância dos Testes Automatizados.** Artigo publicado na **Engenharia de software Magazine**, 1(3), pp. 54-57. 2008. Disponível em: <<http://www.ime.usp.br/~kon/papers/EngSoftMagazine-IntroducaoTestes.pdf>> Acesso em 20 de março de 2013.

BUENO, C.F.S; Campelo, G.B. **Qualidade de software.** Universidade Federal de Pernambuco. Disponível em: <<http://www.cin.ufpe.br/~qualisoft/documentos/diversos/quality.doc>>. Acesso em 30 de março de 2013

COMPUTEWORLD, **EUA perdem 60 bilhões com bugs de software por ano,2002.** International Data Group, Inc. Disponível em <<http://computerworld.uol.com.br/negocios/2002/06/25/idgnoticia.2006-05-15.2695040115/>>. Acesso em 20 de fevereiro de 2013

CORREA, R.A. **Métricas de teste de software (EDD e ERD).**2008. Disponível em: <<http://www.testexpert.com.br/?q=node/1084>>. Acessado em 15 de abril de 2013

DELEPOSTE, F. **Aprendendo TestComplete.** 2011. Disponível em <<http://testesoftware.wordpress.com/2011/07/18/aprendendo-test-complete-i/>>. Acessado em: 29 de julho de 2013.

MYERS, Glenford; WILEY, John. **A arte do software**, 2, Nova Jérsei: 2004.

MARINS, A. F. **Utilizando Selenium IDE.** 2009. Disponível em: <<http://imasters.com.br/artigo/13317/desenvolvimento/utilizando-a-selenium-ide/>>. Acesso em 13 de março de 2013

NIST, *National Institute of Standards and Technology*. Agency of the UIS Department of Commerce Disponível em: <<http://www.nist.gov/index.html>>. Acesso em 20 de janeiro de 2013

PRESSMAN,R.S. **Engenharia de software 6° Edição**. McGraw-Hill, 2006

PRESSMAN, R. S.; LOWE D. **Engenharia Web** – Rio de Janeiro: LTC, 2009.

PRESSMAN,R.S. **Engenharia de software: Uma abordagem profissional**.7° ed. Porto Alegre, McGraw-Hill, 2011.

SHNEIDERMAN, B., **software Psychology: Human Factors in Computer and Information Systems**. 1980. Publicação Winthrop.

Selenium IDE. Disponível em: <<http://docs.seleniumhq.org/projects/ide/>>. Acessado em 15 de julho de 2013

SOMMERVILLE, Ian. **Engenharia de software. 9° Ed.**São Paulo: Pearson Prentice Hall, 2011.

MENDES, E.; MOSLEY, N. **Software Engineering**. University of Auckland, 2006.

MURUGESAN, S.; GINIGE, A. **Web Engineering Introduction and Perspective**. University of Western Sydney, Australia, 2005.

ALKHATIB, G.; RINE, D. **Web Engineering Advancements and Trends: Building New Dimensions of Information Technology**. Information Science reference. New York, 2010

ANEXO 1

Questionário Meditec - www.meditec.net.br[Edit this form](#)

Este questionário é parte do estudo sobre usabilidade de um trabalho de conclusão de curso sobre testes em WebApps. A correta participação do preenchimento deste questionário é de insuma importância para coleta dos dados. As perguntas foram baseadas no livro Engenharia de Software - Uma abordagem Profissional 7ª Edição do Autor Roger S. Presmann

Preencha conforme a numeração:

- 1 - Muito Ruim
- 2 - Ruim
- 3 - Regular
- 4 - Bom
- 5 - Muito Bom

Caso tenha alguma observação ou sugestão, informe logo abaixo de cada pergunta

1- Em relação ao layout: Os mecanismos de navegação, conteúdo e funções são colocados de maneira em que o usuário encontre rapidamente o que procura?

1 2 3 4 5



2 - Em relação a clareza, os textos presente no site são de fácil entendimento?

1 2 3 4 5



3 - Em relação a estética: A cor, tipo de letra, imagens, e características relacionadas facilitam o uso? os usuários se sentem "confortáveis" ao navegar?

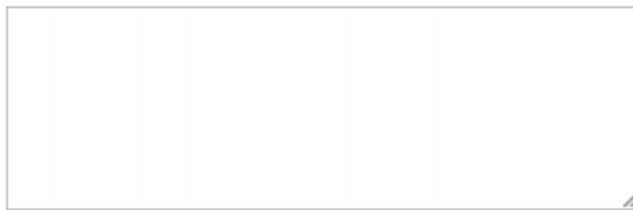
1 2 3 4 5





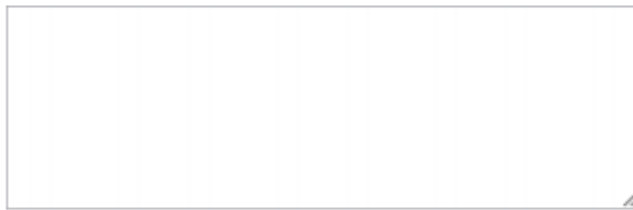
4 - A WebApp otimiza o uso do tamanho da tela e da resolução?

1 2 3 4 5



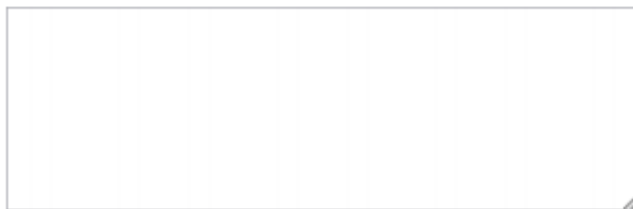
5 - Em relação ao desempenho, o tempo de resposta é aceitável?

1 2 3 4 5



6 - Achou facilmente tudo que queria?

1 2 3 4 5



7 - O formulário de cadastro está adequado à finalidade do site?

1 2 3 4 5



8 - Os objetivos do site estão claros e foram reconhecidos rapidamente?

1 2 3 4 5



9 - Ao se utilizar navegadores diferentes para o acesso nota-se alguma diferença? (Ex. Entre Firefox , Google Chrome ou Internet Explorer)"

- Sim
 Não

10 - Quanto a interface, todos os links funcionam corretamente? Se não descreva quantos(e quais) não funcionam

- Sim
 Não

11 - Existe suporte para pessoas com necessidades especiais?

- Sim
- Não
- Não sei/Não identifiquei

Submit

Never submit passwords through Google Forms.

Powered by
Google Drive

This content is neither created nor endorsed by Google.

[Report Abuse](#) - [Terms of Service](#) - [Additional Terms](#)