UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

COORDENAÇÃO DE LICENCIATURA EM INFORMÁTICA

DESENVOLVIMENTO DE SISTEMAS PARA INTERNET E DISPOSITIVOS MÓVEIS

ROBERTO LUIS PEGORARO

AUTOMAÇÃO DE TESTE EM WEBSITES UTILIZANDO WATIR

MONOGRAFIA DE ESPECIALIZAÇÃO

FRANCISCO BELTRÃO

2014

ROBERTO LUIS PEGORARO

AUTOMAÇÃO DE TESTE EM WEBSITE UTILIZANDO WATIR

Monografia de Especializaçãoapresentada a Coordenação de Licenciatura em Informática, da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de “Especialista em Desenvolvimento de Sistemas para Internet e Dispositivos Móveis”.

Orientador: Prof. Me. Gustavo Yuji Sato

FRANCISCO BELTRÃO

2014

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Francisco Beltrão
Diretoria de Pesquisa e Pós-Graduação
Curso de Especialização em Desenvolvimento de Sistemas para Internet e Dispositivos Móveis

2.

TERMO DE APROVAÇÃO

Dia 17 do mês de Fevereiro de 2015 às: ???? horas, na sala ???? do Câmpus Francisco Beltrão, realizou-se a apresentação pública da monografia pelo estudante Roberto Luis Pegoraro, intitulado: “Automação de Testes em WebSite Utilizando Watir.” Finalizada a apresentação e arguição, a Banca Examinadora declarou **aprovada** a monografia do estudante, requisito parcial para obtenção do título de Especialização em Desenvolvimento e Sistemas para Internet e Dispositivo Móveis.

_____
Professor Gustavo Yuji Sato - UTFPR
(Orientador)

_____
Professor Edson dos Santos Cordeiro - UTFPR
(Convidado)

_____
Professor Marcos Mincov Tenório - UTFPR
(Convidado)

_____
Professor Dr. Ademir Roberto Freddo - UTFPR
(Coordenação)

A folha de aprovação com as rubricas encontram-se disponíveis na Diretoria de Pesquisa e Pós-Graduação, UTFPR, Francisco Beltrão.

Dedico este trabalho à minha família,
pelos momentos de ausência.

AGRADECIMENTOS

Certamente estes parágrafos não irão atender a todas as pessoas que fizeram parte dessa importante fase de minha vida. Portanto, desde já peço desculpas àquelas que não estão presentes entre essas palavras, mas elas podem estar certas que fazem parte do meu pensamento e de minha gratidão.

Agradeço ao meu orientador Prof. Me. Gustavo Yuji Sato, pela sabedoria com que me guiou nesta trajetória.

Aos meus colegas de sala.

A Secretaria do Curso, pela cooperação.

Gostaria de deixar registrado também, o meu reconhecimento à minha família, pois acredito que sem o apoio deles seria muito difícil vencer esse desafio.

Enfim, a todos os que por algum motivo contribuíram para a realização desta pesquisa.

Eu denomino meu campo de Gestão do Conhecimento, mas você não pode gerenciar conhecimento. Ninguém pode. O que você pode fazer, o que a empresa pode fazer é gerenciar o ambiente que otimize o conhecimento. (PRUSAK, Laurence, 1997)

RESUMO

Este trabalho tem por objetivo apresentar a atividade de testes e de automação de testes e sua importância no processo de desenvolvimento de software. Apresentar também a automação de testes web utilizando a ferramenta Watir, a fim de melhorar o processo de desenvolvimento e maximizar a qualidade do produto final. Será apresentado nesse trabalho questões como: o que são testes, porque testar, tipos de automação de testes, por que automatizar, instalação e configuração do ambiente de automação de testes, e exemplificar automação de testes utilizando Watir em website. Ao final serão apresentados os resultados obtidos com a utilização desta ferramenta para que seja usada em tomadas de decisões.

Palavras-chave: Automação. Watir. Qualidade.

ABSTRACT

This work aims to present the activity of testing and test automation and its importance in the software development process. Also, it introduces the use oftests automation using the web tool Watir, which improves the development process and maximize the quality of the final product. Questions like: what are tests?,why testing, test automation types, why automating installation and configuration of test automation environment, and explains how to automate tests using Watir website will be presented in this work. At the end the results obtained from the use of this tool to be used in decision making will be presented.

Keywords: Automation. Watir. Quality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

IDE	Integrated Development Environment
HTML	HyperText Markup Language
XP	Extreming Programming
BDD	Behavior Driven Development
TDD	Test Driven Development
IBM	International Business Machines
CMM	Capability Maturity Model
BSD	Berkeley Software Distribuition

SUMÁRIO

1. INTRODUÇÃO

Segundo Maldonado et al. (2007), a técnica de automação de testes é voltada principalmente para melhoria da qualidade, baseando-se fortemente na teoria de teste de software, para aplicar as recomendações dos testes manuais nos automatizados.

Neste trabalho será apresentado a automação de testes de web em forma de testes de regressão. Segundo ROCHA (2014), os testes de regressão geralmente são executados após a correção de algum defeito ou após a adição de uma nova

funcionalidade. Seu objetivo é garantir que nenhum defeito foi acrescentado ao sistema após sua modificação.

Foi escolhida a ferramenta *Watir* para pesquisa e aplicação de testes automatizados para web, devido a ser uma ferramenta de código aberta a qual proporciona uma grande facilidade de criação e manutenção de scripts e suites de testes automatizados, por suportar vários navegadores em diferentes plataformas e por ser uma ferramenta leve.

1.1Problema de pesquisa

De acordo com CAETANO, muitos projetos de automação de testes costumam fracassar porque a alta gerência costuma acreditar que o único investimento necessáriopara a automação de testes é a compra de uma ferramenta. Entretanto, existem muitos outros investimentos necessá- rios, tais como: Contratação ou capacitação de pessoal para realizar a gestão do projeto de automação de testes, contratação ou capacitação de pessoal para projetar/criar os casos de testes automatizados, compra ou melhoria da infra-estrutura/ambiente para atingir os requerimentos mínimos exigidos pela ferramenta, gastos relacionados à

incorporação de testabilidade na aplicação para torná- la mais fácil de testar, gastos relacionados à criação de provas de conceitos ou protótipos.

A grande dificuldade durante a pesquisa e automação utilizando *Watir*, foi o conhecimento limitado a linguagem de programação *Ruby,* a qual é a linguagem utilizada pelo Watir para realizar a criação dos casos de testes automatizados. E também a falta de uma *IDE* de desenvolvimento própria para o *Watir.*

1.2 Objetivos

Estudar sobre a importância da área de testes e área de automação de testes no processo de desenvolvimento de software.

1.2.1 Objetivo Geral

Realizar estudo sobre automação de testes utilizando a ferramenta *Watir* e demonstrar a ferramenta na prática.

1.2.2 Objetivo Específico

Apresentar a importância da área de testes teste e da automação de testes no processo de desenvolvimento de software, a fim de que possa ser utilizado para aprofundamento no assunto e sirva de ajuda em tomada de decisões referentes ao uso da automação de testes.

Demonstrar a instalação e da ferramenta de automação de testes *Watir* e desenvolver casos de testes automatizados para um sistema de login, e cadastro de usuários e validar os resultados e tempo de execuções dos casos automatizados.

1.3 Justificativa

Segundo CARVALHO, através da automação de testes, o tempo de execução do teste ocorre em proporções bem menores em relação ao tempo executado por um processo de testes manual, onde em algumas situações a equipe não tem tempo suficiente para realizar todos os casos de testes planejados.

A automação de testes segundo Neto, exige mais tempo para implementar os testes, mas provêem mais segurança na manutenção e permitem que os testes sejam executados a qualquer instante.

Com isso, esse trabalho de monografia irá contribuir para agregar conhecimentos sobre automação de teste e também sobre a ferramenta que iremos utilizar para realizar a automação de testes web, o Watir.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos. O primeiro deles apresenta as considerações iniciais, o objetivo e a justificativa do trabalho. O Capítulo 2 apresenta

o referencial teórico acerca dos conceitos que norteiam o trabalho. O Capítulo 3 apresenta a ferramenta de automação de testes Watir. No Capítulo 4 estão os materiais e o método. Por fim é apresentada a conclusão, seguida das referências bibliográficas

2. TESTES DE SOFTWARE E A FERRAMENTA WATIR

Neste capítulo serão apresentados conceitos de processos de testes, fases testes, tipos de testes, automação de testes, tipos de automação de testes, metodologias de automação de testes além de exemplos da ferramenta para automação de testes web o Watir.

2.1 CONCEITO SOBRE TESTES DE SOFTWARE

Segundo MYERS (2004) o "teste de software é um processo, ou um grupo de processos, definidos para garantir que um código faz o que ele foi desenhado para fazer, e não faz nada que não foi especificado para fazer". Já RIOS (2006) define que o teste é "verificar se o software é executado de forma controlada e está fazendo o que deveria fazer, de acordo com os seus requisitos, e não está fazendo o que não deveria fazer".

O processo básico de teste pode ser definido de acordo com as seguintes etapas (RIOS,2014):

- **Planejar Testes:** É definir o planejamento de um projeto de teste de software correspondente a um projeto de desenvolvimento. Deve

seguir as regras básicas de gerência de projetos (riscos, escopo, tamanho do projeto, esforço, cronograma, recursos, indicadores, entre outros) e o artefato gerado é o Plano de Teste.

- **Projetar Testes:** Contempla criação dos casos de teste e demais artefatos necessários às atividades de execução dos testes conforme definido no Plano de Teste. Nesta fase são definidos os cenários de teste, elaborados os casos de teste, estruturados os *scripts* de teste e definido o procedimento de teste.
- **Executar Testes:** Executar casos de teste e/ou *scripts* automáticos, que foram definidos para cada iteração, bem como executar testes específicos, como teste de performance e outros. Antes de executar os casos de teste devem ser definidos os responsáveis pela sua execução e acompanhamento de defeitos.
- **Gerenciar Defeitos:** Registrar e acompanhar a correção dos defeitos, decorrentes da execução dos testes. Se for identificado que o defeito persiste, após o re-teste, a correção é rejeitada, caso contrário ela é aprovada. Os defeitos devem ser classificados por prioridade e severidade.
- **Analisar Resultados:** Ao final do projeto deve ser elaborado um relatório contemplando todas as informações relevantes coletadas durante o seu decorrer, assim como problemas ocorridos e não conformidades encontradas. Os indicadores fornecidos pelo projeto devem ser catalogados como histórico para uso em processos de planejamento e de estimativas. As lições apreendidas devem ser usadas na revisão do processo.

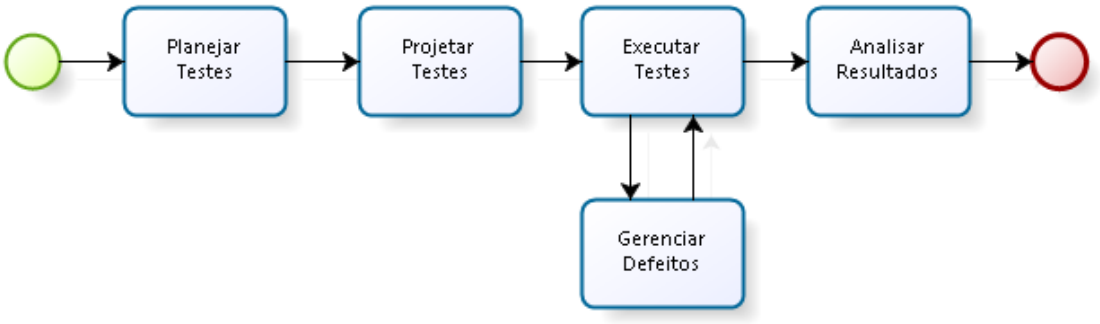


Figura 1 - Processo Básico de Teste
Fonte: RIOS, (2014)

2.2 FASES DE TESTE

O teste de software é dividido em algumas fases dependendo do objetivo inicial do teste, dentre as quais estão: testes unitários, de integração, de sistema e de aceitação. A seguir serão apresentadas brevemente cada uma dessas fases.

2.2.1 Testes Unitários

O objetivo do teste unitário é testar a menor funcionalidade existente do software, ou seja, isolar parte dos códigos e métodos, e analisar se essas funcionalidades obtêm o retorno esperado mediante a um valor informado (SILVA, 2014). De acordo com LOURENÇO (2014) os alvos desse tipo de teste são os métodos dos objetos ou mesmo pequenos trechos de código. Assim, o objetivo é encontrar falhas de funcionamento dentro de uma pequena parte do sistema funcionando independentemente do todo.

2.2.2 Testes de Integração

Na fase de testes de integração são realizados testes com o objetivo de encontrar falhas originadas da integração das unidades já testadas. Não faz parte

desta fase o teste de integração com outros sistemas. Segundo Pierri (2014) um exemplo de teste de integração seria na integração do cadastro de clientes com a função que valida CPF, as duas unidades já foram testadas individualmente na fase de testes de unidade porém é neste momento que a interação entre elas é validada.

2.2.3 Testes de Sistema

Teste de Sistema é uma verificação de que o software atende as especificações feitas pela análise de requisitos, tanto requisitos funcionais como não funcionais. Nesta fase o responsável pelos testes irá utilizar o sistema como um todo, simulando o usuário final, buscando falhas nas funcionalidades. Nesta fase também são testadas integrações com outros sistemas.

2.2.4 Testes de Aceitação

Nesta última fase de testes, os testes são realizados pelo usuário final do sistema, onde serão simuladas as tarefas do dia-a-dia a fim de verificar se o sistema está de acordo com o que foi solicitado. Existem três estratégias comuns para implementar os testes de aceitação. São elas (IBM, 2014):

- **Aceitação formal:** O teste de aceitação formal costuma ser uma extensão do teste de sistema. Seu processo é altamente gerenciado, por isso os testes são planejados e projetados com o mesmo cuidado e nível de detalhe do teste do sistema, seus casos de teste mais relevantes para tal. Em muitas empresas de software o teste de aceitação formal é totalmente automatizado. Este teste pode ser executado pela própria equipe de testes em conjunto com representantes da organização do usuário final, ou inteiramente pela organização do usuário final ou um grupo objetivo de pessoas por ela escolhido.
- **Aceitação informal ou teste alfa:**No teste de aceitação informal, os procedimentos para executar o teste não são definidos com tanto rigor como no teste de aceitação formal, as funções e as tarefas de negócios a serem exploradas são identificadas e documentadas, mas não há casos de teste específicos para seguir. É executado por um

cliente nas instalações do desenvolvedor. O software é usado num ambiente controlado, onde o desenvolvedor acompanha o usuário e registra erros e problemas de uso ocorridos.

- **Teste beta:** O teste beta também é um teste de aceitação voltado para softwares cuja distribuição atingirá grande número de usuários de uma ou várias empresas compradoras. PRESSMAN (2006) afirma que o teste beta é conduzido em uma ou mais instalações do cliente, pelo usuário final do software, onde o desenvolvedor não está presente. O cliente registra os problemas que são encontrados e relata-os para a equipe de desenvolvimento. Com o resultado dos problemas relatados durante os testes beta, a equipe de desenvolvimento faz as modificações e depois se prepara para liberar o software para todos os clientes.

2.3 TIPOS DE TESTES

Para ajudar na fase de testes, existem tipos de testes, que aperfeiçoam os objetivos dos testes e facilitam o trabalho do testador.

Nos próximos tópicos serão apresentados os tipos de testes como testes funcionais utilizando valores de entrada e resultados conhecidos (caixa-preta), testes que permitem visualização do código (caixa-branca), testes que verificam se o sistema está lento (teste de performance), testes que validam se o sistema está difícil de usar (teste de usabilidade).

2.3.1 Testes Funcionais(Caixa-Preta)

Segundo MYERS (2004) o teste de caixa preta trata o software como uma caixa escura cujo conteúdo é desconhecido e da qual só é possível visualizar o lado externo, ou seja, os dados de entrada fornecidos e as respostas produzidas como saída.

Os testes funcionais são efetuados fornecendo dados de entrada, e validando se os resultados obtidos são iguais aos resultados previamente conhecidos. Caso os resultados não forem iguais o teste encontrou uma falha.

Muito se fala sobre a cobertura de testes, o quanto testar, quais e quantas entradas utilizar, e de onde retirar estas entradas. Para isto, PRESSMAN (1992) define que o teste funcional envolve dois passos principais: identificar as funções que o software deve realizar e criar casos de teste capazes de checar se essas funções estão sendo realizadas pelo software.

2.3.2 Testes Estruturais(Caixa-Branca)

A técnica de teste de estrutural é conhecida por vários nomes, tais como, teste de caixa-branca e teste de caixa de vidro. Esta técnica consiste em determinar dados de entrada para análise da lógica do software, onde o desenvolvedor ou testador realizará o teste direto no código fonte do software MYERS (2004).

Está técnica verifica a complexidade lógica do software utilizando a sua estrutura para descobrir os caminhos lógicos do software. O teste é exercido de modo que todas as rotinas sejam testadas via código, avaliando a estrutura interna do software, como funções, condições, fluxo de dados, ciclos, decisões lógicas entre outras.

Portanto, o teste estrutural é projetado com base na estrutura interna do sistema. Este tipo de teste é desenvolvido para permitir uma verificação mais precisa do funcionamento do software de forma que seja feita uma análise do código-fonte, visando complementar a técnica funcional e informações obtidas pela aplicação, levando em conta os critérios considerados como relevantes nas atividades de manutenção, depuração e para confiabilidade do software. Sendo assim, é recomendada para as fases de teste de unidade e teste de integração, cuja responsabilidade principal fica a cargo dos desenvolvedores do sistema, que por sua vez conhecem bem o código produzido (MOREIRA, 2014).

2.3.3 Testes de Performance

É projetado para testar o desempenho do software durante a execução de uma aplicação, visando descobrir situações que levem a uma possível falha do software. É um tipo de teste destinado a determinar o tempo de resposta, a

confiabilidade e a escalabilidade da aplicação sob uma determinada carga de trabalho (ELIZA, 2014).

Antes de iniciar o teste de *performance*, é preciso ter pelo menos um objetivo bem definido sobre o que se espera como resposta, por exemplo, se um *site* de e-*commerce* lançar uma promoção em seus produtos e esperar obter um acesso simultâneo de 10 mil compradores, o teste de *performance* utilizando-se de ferramentas como o *JMeter* deve garantir que com esta quantidade de acessos simultâneos o seu *e-commerce* não irá perder *performance* no tempo de resposta das transações.

Para SOUSA (2014) há alguns tipos de testes de *performance*, entre eles:

- **Teste de desempenho:** consiste no teste utilizado para verificar o desempenho do sistema num cenário previsto de baixa ou média carga. Através dele é possível mensurar o tempo de resposta ao acionar os comandos disponíveis e obter informações a respeito dos recursos físicos necessários num cenário comum de funcionamento.
- **Teste de carga:** utilizado para verificar o desempenho do software num cenário onde há grande volume de usuários com acesso simultâneo ao sistema, num horário de pico, por exemplo.
- **Teste de *stress*:** é o teste utilizado para verificar o limite máximo do sistema submetido a um grande volume de acesso simultâneo, portanto visa identificar o ponto em que o sistema deixa de atender ao mínimo especificado. É vantajoso pois fornece informações que permitem evoluir a arquitetura do software. Os indícios de que o sistema não atende mais ao esperado podem se manifestar de diversas formas, as mais comuns são:
    - Os dados deixam de ser salvos ou são corrompidos;
    - Ao retornar às condições normais de funcionamento, alguns recursos permanecem degradados;
    - A aplicação não atende a determinados comandos;
    - A aplicação trava/deixa de funcionar definitivamente.

2.3.4 Testes de Usabilidade

Usabilidade envolve critérios objetivos, como tempo e quantidade de operações necessárias para executar uma tarefa e a frequência de erros do usuário, além da subjetiva satisfação geral do mesmo (PEZZÈ 2005).

PEZZÈ (2005) complementa que, mesmo que a usabilidade seja apoiada na percepção do usuário e, assim, baseada em seu retorno, ela pode ser verificada cedo no projeto e durante todo o ciclo de desenvolvimento. O processo de verificar e validar a usabilidade inclui os seguintes passos principais:

- **Inspecionar as especificações** usando listas de verificação de usabilidade. A inspeção provê um retorno antecipado da usabilidade.
- **Testar primeiro os protótipos** com os usuários finais para explorar seu modelo mental (teste exploratório), avaliar alternativas e validar a usabilidade do software.
- **Testar as versões de maneira incremental** tanto com especialistas em usabilidade como com usuários finais para monitorar o progresso e antecipar problemas de usabilidade.
- **Executar testes de sistema e aceitação** que incluam inspeções e testes usando especialistas, testes com usuários finais, testes de comparação entre competidores e análises e verificações normalmente feitas de maneira automática, como teste de conectividade e de compatibilidade com navegadores.

2.3.5 Testes de Regressão

Testes de regressão são aplicados a cada nova versão ou entrega do software. Este tipo de testes visa basicamente garantir que o que já estava funcionando no sistema, continue funcionando, ou seja, valida se as novas regras aplicadas no sistema não afetaram o que já estava funcionando.

De acordo com ROCHA (2014), os testes de regressão geralmente são executados após a correção de algum defeito ou após a adição de uma nova funcionalidade.

Como os testes regressivos geralmente são os mesmos, e a cada versão é necessário a execução dessa massa de testes para garantir o funcionamento, muitas empresas optam pela automação destes casos de testes, para que o testador foque seus esforços em outros tipos de testes. Com a massa de dados de testes de regressão automatizados é possível executar os testes a cada nova versão ou entrega do software com mais rapidez, confiabilidade e menor custo.

Segundo Bartié (2002), O risco de que as novas alterações tenham comprometido as funcionalidades anteriores tendem a aumentar ainda mais à medida que o software vai se tornando mais complexo. E Bartié (2002) ainda complementa, os custos relativos à execução dos testes de progressão não são importantes. São importantes os custos da execução dos testes de regressão, pois esses devem ser continuamente executados ao longo da vida do software.

2.4 METODOLOGIAS DE DESENVOLVIMENTO ÁGIL

Atualmente muito se fala em desenvolvimento ágil, pois os clientes exigem as suas soluções o mais rápido possível, e conceber um software com toda a burocracia do processo tradicional de desenvolvimento é demorado e na maioria das vezes deixa a desejar no quesito tempo de entrega. Com isto surgiram algumas metodologias de desenvolvimento ágil como *Scrum* e *eXtreme Programing* (XP), que são voltadas a entregar o que foi proposto o mais rápido possível, porém sem ter um grande foco na documentação do sistema, conforme realizado em um modelo tradicional de desenvolvimento, mas apenas definindo alguns artefatos considerados indispensáveis ao processo, o que pode comprometer de alguma forma a qualidade do processo. Pensando nisto surgiram algumas metodologias de desenvolvimento ágil voltadas para a qualidade do software, tais como o Desenvolvimento Orientado a Teste (TDD) e Desenvolvimento Orientado a Comportamento (BDD), que serão explicadas a seguir.

2.4.1 Desenvolvimento Orientado a Testes(Test Diven Development – TDD)

De acordo com (BECK, 2003) TDD é uma abordagem evolutiva para o desenvolvimento onde é possível escrever um teste antes de escrever código de produção suficiente para cumprir esse teste. Ou seja, é feito primeiro um caso de um teste unitário, depois é desenvolvido o código do sistema para fazer aquele teste passar. Por isto o maior objetivo do TDD é desenvolver o código mais simples e limpo para atender aquele caso de teste.

A Figura 4 mostra como funciona o fluxo de desenvolvimento em TDD.

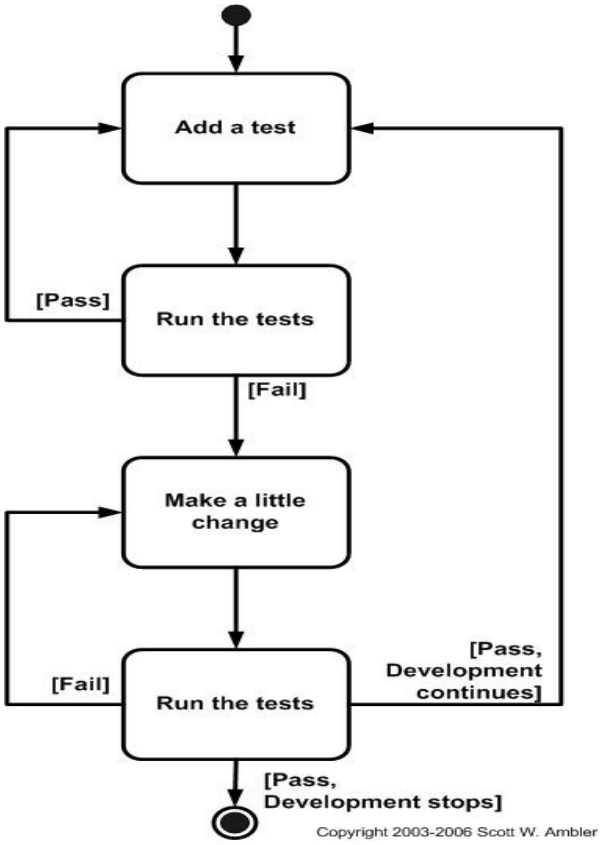


Figura 2 - Fluxo TDD
Fonte: AMBLER (2014)

Um teste é adicionado, em seguida o código do sistema é desenvolvido e o teste é executado. Se passou, adiciona outro teste para continuar o desenvolvimento, se falhou, faz uma pequena mudança e executa o teste novamente mantendo este ciclo até o teste passar.

2.4.2 Desenvolvimento Orientado a Comportamento(Behavior Driven Development - BDD)

De acordo com SOARES (2014) BDD é uma técnica de desenvolvimento ágil evoluída do TDD, que visa integrar regras de negócios (estórias) com linguagem de programação.

Para que uma estória seja considerada completa ela deve ter alguns requisitos necessários, estes são chamados de critérios de aceite. Não existe limites de critérios de aceite nem regras de como eles devem ser descritos, mas o

fundamental é que todos eles representem um comportamento testável que agregue valor para o cliente.

Exemplo de estória:

Funcionalidade: <descrição da funcionalidade>

Como um <usuário/ator>
Eu quero <meta a ser alcançada>
De modo que <a razão para alcançar a meta>

Um exemplo de critério de aceite pode ser descrito desta forma:

Cenário: <descrição do teste>
Dado <um estado conhecido>
Quando <um determinado evento ocorre>
Então <isso deve ocorrer>

A Figura 5 mostra um exemplo de história e critérios de aceite feito por Ribeiro (2014).

```
#Language: pt
  Funcionalidade: Leitor de tipos de Triângulo

  Para conhecer o tipo de um triângulo
  Como um aluno da matemática
  Eu quero informar os tamanhos do lado de um triângulo e saber qual o tipo do triângulo

  NARRATIVA

  Um triângulo com todos os lados iguais é chamado Equilátero
  Um triângulo com dois lados iguais é chamado Isósceles
  Um triângulo com todos os lados diferentes é chamado Escaleno

  FORA DE ESCOPO

  -Validar triângulos inválidos
  -Exibir o triangulo graficamente
  -validação de entrada de dados do usuário

  Cenario: Consultando um triângulo escaleno
    Dado que estou na página de consulta de triângulos
    Quando eu informo os lados de um triangulo:
      | lado_a | lado_b | lado_c |
      |   3    |   4    |   5    |
    Entao o sistema informa que o triângulo é "Escaleno"

  Cenario: Consultando um triângulo equilátero
    Dado que estou na página de consulta de triângulos
    Quando eu informo os lados de um triangulo:
      | lado_a | lado_b | lado_c |
      |   5    |   5    |   5    |
    Entao o sistema informa que o triângulo é "Equilátero"

  Cenario: Consultando um triângulo isósceles
    Dado que estou na página de consulta de triângulos
    Quando eu informo os lados de um triangulo:
      | lado_a | lado_b | lado_c |
      |   5    |   4    |   5    |
    Entao o sistema informa que o triângulo é "Isósceles"
```

Figura 3 - Estória e Critérios de Aceite
Fonte: RIBEIRO (2014)

Estes cenários listados serão utilizados como base para codificação do sistema e servem como uma documentação “viva” do software.

2.5 AUTOMAÇÃO DE TESTES

Em um ambiente de desenvolvimento de software, frequentemente há a necessidade de execução de testes regressivos, ou seja, re-teste de uma ou mais funcionalidades a fim de identificar defeitos inseridos por novas funcionalidades ou para correção de defeitos. Quando este processo ocorre de forma manual, a equipe de testes passa a gastar mais tempo em testes regressivos do que em testes de novas funcionalidades, e consequentemente, deixa de acompanhar a demanda da equipe de desenvolvimento ao longo do tempo, atrasando entregas ou, mesmo, entregando o produto sem ter sido completamente testado.

A automação de testes tem o objetivo de reduzir o envolvimento humano em atividades manuais repetitivas(CAETANO). Desta forma, quando utilizada corretamente, permite que testes regressivos sejam executados a qualquer momento e quantas vezes forem necessárias. Outras vantagens oferecidas pela automação são a rapidez na execução dos testes comparada ao manual, e também ser menos suscetível a erros “humanos”, visto que a interpretação dos casos de teste é sempre a mesma.

Segundo Caetano (2008), “os tipos de automação são normalmente agrupados de acordo com a forma como os testes automatizados interagem com a aplicação”, dois dos tipos de automação de testes regressivos mais importantes são os testes automatizados baseados em interface gráfica e os testes automatizados dirigidos a dados.

Os testes automatizados baseados em interface gráfica são realizados com base na interface da aplicação, onde a ferramenta de automação fornece um recurso de captura de ações do usuário enquanto este utiliza a aplicação. A ferramenta transforma essas ações em *scripts* que podem ser reproduzidos posteriormente. Este tipo de teste automatizado tem a desvantagem de ser dependente da interface, pois, como explica Caetano (2008), “se a interface mudar, os testes falham”.

Nos testes automatizados dirigidos a dados (*Data-Driven*) a interface fornece a entrada dos dados e apresentação de resultados, onde os testes executam a mesma ação várias vezes, porém com dados diferentes. Este paradigma tem a

vantagem de estar focado na camada onde existe maior probabilidade de existir erros, porém também possui dependência da interface.

Com a criação de testes automatizados dirigidos a dados, os dados não são mais constantes nos *scripts*, e começa-se a utilizar o mecanismo *Test Datapool*. O *Test Datapool* irá armazenar os dados a serem utilizados na execução de um mesmo *script*, reutilizando-o. Desta forma, cada *script* precisará ser capturado apenas uma vez, e em seguida trocam-se os dados constantes pela referência dos dados contidos no *Test Datapool.*

A automação é um investimento a longo prazo com retorno garantido quando bem implementada. Precisa ser considerada como um projeto como todos os outros, com planejamento detalhado e demais atividades de projetos. Caetano (2008, p.46) comenta que "muitos projetos de automação de testes costumam fracassar porque a alta gerência costuma acreditar que o único investimento necessário para a automação de testes é a compra de uma ferramenta", porém existem outros investimentos necessários, como: contratação e capacitação de pessoal,compra ou melhoria de infraestrutura e ambiente para atender aos requisitos da ferramenta, gastos para tornar a aplicação mais fácil de testar (testabilidade), entre outros.

Os testes automatizados podem ser definidos como uma estratégia para reduzir o envolvimento humano em atividades manuais repetitivas, porém não devem ser considerados como substitutos para os testes manuais. Muitas empresas decidem iniciar a automação de testes mesmo sem possuir uma maturidade mínima em seu processo de testes. Como cita Caetano (2008, p. 42), segundo o criador do *Capability Maturity Model*(CMM), Watts S. Humphrey, "a qualidade do produto final é diretamente proporcional à qualidade do processo utilizado no seu ciclo de vida". Logo, o sucesso da automação de testes depende de testes manuais maduros e consistentes - não é possível obter resultados positivos nos testes automatizados, sendo que estes foram baseados em testes manuais errados, ambíguos ou inconsistentes. Afinal, "se você automatizar o caos,terá o caos automatizado".

3. FERRAMENTA DE TESTES

Esse capítulo apresenta a ferramenta *Watir*, escolhida para apresentação neste trabalho, por possuir poucos materiais, principalmente em português desta ferramenta. Sendo assim, a seguir será apresentada a ferramenta que vem sendo utilizada por organizações da área que se utilizam da automação de testes.

3.1 WATIR

O Watir é um open-source (BSD) da família de bibliotecas Ruby para a automatização de navegadores web. Ele permite escrever testes que são fáceis de ler e manter. É simples e Flexível.(WATIR, 2014)

Watir é utilizado em automação de testes funcionais , onde são feitos scripts de testes esperando os devidos resultados, ou realizando específicas ações, para garantir a qualidade e funcionalidade de um website ou aplicativo web. Como é uma biblioteca Ruby, logo a linguagem de programação utilizada para implementar os scripts de testes é o Ruby.

Vantagens em utilizar o Watir (WATIR, 2014):

- É uma ferramenta livre open-source. Não há custos para usar a ferramenta.
- Há uma comunidade ativa e crescente por trás dele.
- Utiliza Ruby, uma linguagem de script moderna e cheia de recursos.
- Suporta vários navegadores em diferentes plataformas.
- É poderoso e fácil de usar, e muito leve.

3.1.1 Utilizando o Watir

Pré-Requisitos para utilização:

- Possuir a última versão do Ruby instalado. (RUBY)
- Possuir o Sublime Text instalado. (SUBLIME)
- Mapear a variável de ambiente GEM_HOME para o local onde está instalado o Ruby.
- Adicionar na variável PATH o caminho %GEM_HOME%\bin

- Instalar as bibliotecas necessárias para funcionamento do Watir, para isso basta abrir o CMD(Windows) e digitar os comandos a seguir:
- gem install watir-webdriver
- gem install test-unit
- gem install ruby_gem

3.1.2 Utilizando a Ferramenta Watir para Automatizar uma Aplicação Web

O script abaixo demonstra a automação de uma aplicação web onde será aberto o navegador, acessado a página da aplicação, adicionado os valores correspondentes aos lados de um triângulo, verificada a saída dos dados, e por fim irá fechar o navegador.

TesteTriangulo.rb | SuiteTriangulo.rb

```ruby
require 'rubygems'
require 'test/unit'
require 'watir-webdriver'

class TesteTriangulo < Test::Unit::TestCase
    class << self #Define como singleton os métodos 'startup' e 'shutdown'
        def startup
            #Instancia um novo navegador
            $browser = Watir::Browser.new :firefox
            #Maximiza o browser
            $browser.window.maximize
        end

        def shutdown
            $browser.close
        end
    end

    #para ser um teste é necessário iniciar com test, tudo minúsculo
    def test_Triangulo_Equilatero
        acessa_pagina

        digita_lado1(1)
        digita_lado2(1)
        digita_lado3(1)

        clica_calcular

        assert_equal 'Triângulo Equilatero', $browser.p(:class, 'mensagem').text
    end

    #para ser um teste é necessário iniciar com test, tudo minúsculo
    def test_Triangulo_Isosceles
        acessa_pagina

        digita_lado1(1)
        digita_lado2(2)
        digita_lado3(1)

        clica_calcular

        assert_equal 'Triângulo Isosceles', $browser.p(:class, 'mensagem').text
    end

    #para ser um teste é necessário iniciar com test, tudo minúsculo
    def test_Triangulo_Escaleno
        acessa_pagina

        digita_lado1(1)
        digita_lado2(2)
        digita_lado3(3)

        clica_calcular

        assert_equal 'Triângulo Equilatero', $browser.p(:class, 'mensagem').text
    end

    def acessa_pagina
        $browser.goto('http://triangulos-1.herokuapp.com/')
    end

    def digita_lado1(lado1)
        $browser.text_field(:id, 'triangulo_lado1').set(lado1)
    end

    def digita_lado2(lado2)
        $browser.text_field(:id, 'triangulo_lado2').set(lado2)
    end

    def digita_lado3(lado3)
        $browser.text_field(:id, 'triangulo_lado3').set(lado3)
    end

    def clica_calcular
        $browser.button(:name, 'commit').click
    end
end
```

Figura 4–Script Automatizado Usando Watir

Após a execução é possível analisar os resultados referentes a execução automatizada como mostra na figura abaixo:

```
Loaded suite C:/Users/robertopegoraro/Desktop/TesteTriangulo
Started
.F
===============================================================================
Failure:
test_Triangulo_Escaleno(TesteTriangulo)
C:/Users/robertopegoraro/Desktop/TesteTriangulo.rb:55:in `test_Triangulo_Escaleno'
     52:
     53:         clica_calcular
     54:
  => 55:         assert_equal 'Triângulo Equilatero', $browser.p(:class, 'mensagem').text
     56:     end
     57:
     58:     def acessa_pagina
<"Tri\u00E2ngulo Equilatero"> expected but was
<"Tri\u00E2ngulo Escaleno">

diff:
? Triângulo Equilatero
?            sca    n
===============================================================================
.

Finished in 27.812 seconds.

3 tests, 3 assertions, 1 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
66.6667% passed

0.11 tests/s, 0.11 assertions/s
[Finished in 28.7s with exit code 1]
```

Figura 5–Resultado da execução de um caso automatizado

Durante a execução deste script de teste, foi forçado um erro, onde o teste esperava um Triângulo Equilátero, mas a aplicação retornou Triângulo Escaleno. É possível perceber isso na parte destacada em vermelho da imagem.

No ciclo de vida de desenvolvimento é necessário rodar diversas vezes os roteiros automatizados, e para isso são agrupados todos os testes automatizados em suítes para facilitar essa execução. Na figura a seguir demonstra como é montada uma suíte de testes automatizados.

TesteTriangulo.rb | SuiteTriangulo.rb

```
1  require 'test/unit/testsuite'
2  require 'test/unit/ui/console/testrunner'
3
4  #Nome do Arquivo onde contém os testes
5  require_relative 'TesteTriangulo'
6
7
8  class TS_TestesTriangulo
9    def self.suite
10     suite = Test::Unit::TestSuite.new
11     #Nome da classe onde contém os testes
12     suite << TesteTriangulo.suite
13     return suite
14   end
15 end
16
17 Test::Unit::UI::Console::TestRunner.run(TS_TestesTriangulo)
```

Figura 6–Suíte de testes automatizados

3.1.3 Comandos e Métodos Suportados

Existem diversos comandos que podem ser utilizados na automação de testes utilizando o Watir, para isso precisamos saber os métodos suportados por elementos, como mostra a tabela abaixo:

button	<input> tags with type=button, submit, image or reset
radio	<input> tags with the type=radio; known as radio buttons
check_box	<input> tags with type=checkbox
text_field	<input> tags with the type=text (single-line), type=textarea (multi-line), and type=password
hidden	<input> tags with type=hidden
select_list	<select> tags, known as drop-downs or drop-down lists
label	<label> tags (including "for" attribute)
span	tags
div	tags
p	<p> (paragraph) tags
link	<a> (anchor) tags

table	<table> tags, including row and cell methods for accessing nested elements
image	<img> tags
Form	<form> tags
Frame	frames, including both the <frame> elements and the corresponding pages
Map	<map> tags
Area	<area> tags
Li	tags
h1 - h6	<h1>, <h2>, <h3>, <h4>, <h5>, <h6> tags

Tabela 1–Elementos HTML atualmente suportados pelo Watir
Fonte: PETTICHORD(2014).

Agora que já foram os elementos suportados, é necessário saber quais as formas que podem ser utilizadas pela ferramenta Watir para interagir com esses elementos. Estas formas, são os atributos dos elementos HTML e podem ser visualizados na tabela a seguir.

:id	Used to find an element that has an "id=" attribute. Since each id should be unique, according to the XHTML specification, this is recommended as the most reliable method to find an object. *
:name	Used to find an element that has a "name=" attribute. This is useful for older versions of HTML, but "name" is deprecated in XHTML. *
:value	Used to find a text field with a given default value, or a button with a given caption, or a text field
:text	Used for links, spans, divs and other element that contain text.
:index	Used to find the nth element of the specified type on a page. For example, button(:index, 2) finds the second button. Current versions of WATIR use 1-based indexing, but future versions will use 0-based indexing.
:class	Used for an element that has a "class=" attribute.
:title	Used for an element that has a "title=" attribute.
:xpath	Finds the item using xpath query.
:method	Used only for forms, the method attribute of a form is either GET or POST.
:action	Used only for form elements, specifies the URL where the form is to be submitted.

:href	Used to identify a link by its "href=" attribute.
:src	Used to identify an image by its URL.

Tabela 2–Formas de identificação de Elementos HTML.
Fonte: PETTICHORD(2014).

Após saber quais elementos são suportados pela ferramenta, e também quais as formas de capturar esses elementos, é necessário saber quais formas são usadas para determinado elemento, pois nem todas as formas podem ser utilizadas para todos os elementos. Na tabela a seguir é apresentada as possíveis formas para determinado atributo.

	:id	:name	:value	:text	:caption	:index	:class	:xpath	:title	:method	:action	:href	:src	:for	multiple attribute support?
button	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
radio	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
check_box	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
text_field	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
hidden	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗
select_list	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗
label	✓	✓	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓
span	✓	✓	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓
div	✓	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓
p	✓	✓	✓	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓
link	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✗	✗	✓
table	✓	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓
image	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✗	✓
form	✓	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗	✗	✓
frame	✓	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✓	✗	✓
map	✓	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓
area	✓	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓
li	✓	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓
h1 - h6	✓	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓

Tabela 3– Métodos suportados por Elemento
Fonte: PETTICHORD(2014).

4. MATERIAIS E METODOS

Este capítulo apresenta os materiais e o método utilizados para a realização deste trabalho.

4.1 MATERIAIS

A realização desse trabalho baseou-se basicamente na exploração de materiais bibliográficos (ex. livros, artigos, sites) acerca dos conceitos, tipos e técnicas de teste. Seguindo da exploração das ferramentas de testes aqui apresentadas, visando disponibilizar ao leitor desse trabalho os conhecimentos necessários para obtê-las e iniciar sua utilização.

4.2 MÉTODOS

Quanto à natureza da pesquisa, esse trabalho desenvolveu uma **pesquisa aplicada**, a qual objetiva gerar conhecimentos para uma futura aplicação prática, dirigidos à solução de problemas específicos, nesse caso, a realização de uma prática de automação de testes.

Considerando os objetivos, foi realizada uma **pesquisa exploratória**: visando proporcionar maior familiaridade com o problema. Segundo GIL, esse tipo de pesquisa envolve levantamento bibliográfico.

Sendo assim, como técnicas de pesquisa foram utilizadas a **pesquisa bibliográfica e documental** (GIL), pois procurou apresentar aspectos importantes da área de teste de software e o processo de automação dessa atividade, além de apresentar ferramentas para automação de diversos tipos de testes, destacando suas características e formas de obtenção.

5. AUTOMATIZANDO UM WEBSITE UTILIZANDO WATIR

Para atender os objetivos deste trabalho, será desenvolvida a automação de uma página web utilizando a ferramenta Watir. Como base para os testes será utilizado cenários de testes escrito em formato de BDD. A página web contém, um login, uma tela de listagem de usuários, uma tela de cadastro, e uma tela de alteração.

A primeira etapa para automatizar este website é criar a estrutura básica de um script de teste para a ferramenta Watir.

```
require 'rubygems'
require 'test/unit'
require 'watir-webdriver'

class TesteTriangulo < Test::Unit::TestCase
   class << self #Define como singleton os métodos 'startup' e 'shutdown'
      def startup
         #Instancia um novo navegador
         $browser = Watir::Browser.new :chrome
         #Maximiza o browser
         $browser.window.maximize
      end

      def shutdown
        #  $browser.close
      end
   end
```

Criada a estrutura básica, iremos iniciar a automação seguindo o cenário de testes número 1:

Funcionalidade: Cadastro de usuário.

Cenário 1: Efetuar login.

Dado que estou na tela de login.

Quando informar o usuário Admin.

E informar a senha Admin.

E clicar no botão Efetuar Login.

Então deve logar e Exibir a listagem de usuários

Figura 6–Tela de Login

Para automatizar o login, iremos utilizar o seguinte método de teste:

```
def test01Login
        $browser.goto('http:///robertopegoraro/login.html')
        $browser.text_field(:id, 'user').set('admin')
        $browser.text_field(:id, 'password').set('admin')
        $browser.button(:id, 'submit').click
End
```

Após realizar o login, é apresentada a tela de listagem de usuários. A partir desta tela é possível realizar algumas ações, como cadastrar, alterar e excluir.

Logado com usuário: Admin Sair

Cadastrar

ID	Nome	Sobrenome	Email	Ações	
1	Roberto	Pegoraro	roberto_luis_15@hotmail.com	Alterar	Excluir
3	ADALBERTO	GASQUES	GASQUESo@teste	Alterar	Excluir
4	ADALBERTO GOMES	SANTOS	SANTOS@teste	Alterar	Excluir
5	ADALTO	RAMOS	RAMOS@teste	Alterar	Excluir
6	ADAO ASSIS	OLIVEIRA	OLIVEIRA@teste	Alterar	Excluir
7	ADEBAR GARCIA	SILVA	SILVA@teste	Alterar	Excluir
8	ADELIDE	PINHEIRO	PINHEIRO@teste	Alterar	Excluir
9	ADELIO	TONIOLO	TONIOLO@teste	Alterar	Excluir
10	ADEMIR	SOUZA	SOUZA@teste	Alterar	Excluir
11	ADEVAR	MARQUES	MARQUES@teste	Alterar	Excluir
12	ADILSON	TELES	TELES@teste	Alterar	Excluir

Figura 7–Listagem de Usuários

Efetuado o login, e estando na tela de listagem de usuários, é possível realizar o segundo cenário de testes.

Cenário 2: Cadastrar um novo usuário.

Dado que estou na tela de cadastro de Usuário.

Quando eu informar um nome.

E eu informar um Sobrenome.

E eu informar um Email.

E eu clicar no botão Cadastrar.

Então deve exibir a mensagem "Usuário Cadastrado com sucesso!"

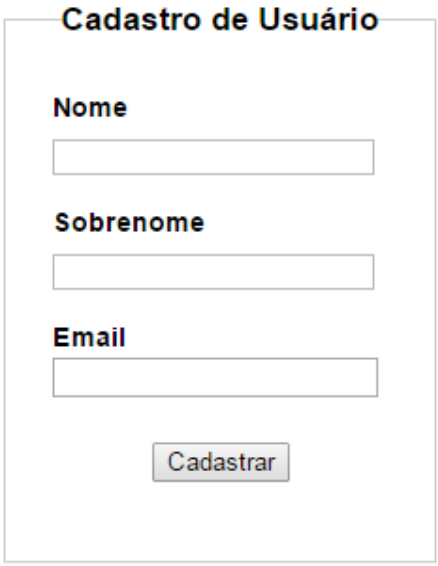


Figura 8–Cadastro de Usuários

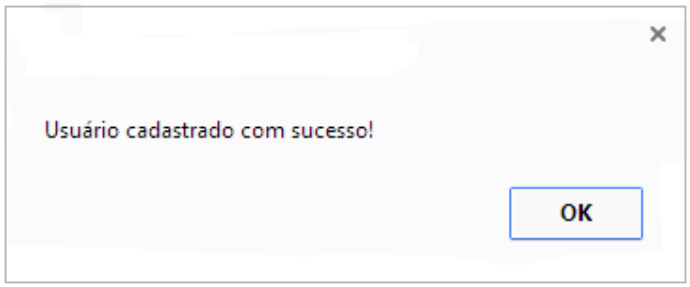


Figura 9–Mensagem de Confirmação

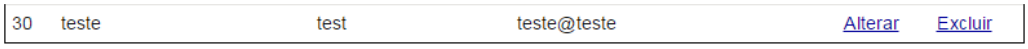


Figura 10–Usuário sendo listado.

Para realizar a automação do BDD e validar a mensagem de gravado com sucesso, e também validar na listagem se o usuário foi realmente inserido, iremos utilizar o seguinte código:

```
def test02_CadastrarUsuario
    $browser.link(:text, 'Cadastrar').click
    $browser.text_field(:name, 'nome').set('teste')
```

```
    $browser.text_field(:name, 'sobrenome').set('test')
    $browser.text_field(:name, 'email').set('teste@teste')
    $browser.button(:name, 'Cadastrar').click

    assert_equal("Usuário cadastrado com sucesso!", $browser.alert.text)
    $browser.alert.ok

    row = $browser.table.tbody.trs.find do |tr|
      tr.td(:index => 3).text == 'teste@teste'
    end

    #valida se encontrou o email na lista
    assert_equal(6, row.cells.lenght)
end
```

Realizado o cadastro do usuário, será então criado a automação do terceiro cenário, a alteração de um usuário:

Cenário 3: Alterar um usuário.

Dado que estou na tela de alteração de Usuário.

Quando eu alterar o Nome.

E alterar o Sobrenome.

E clicar no botão Alterar.

Então deve exibir a mensagem "Usuário Alterado com sucesso!"

Figura 11–Alteração de Usuários

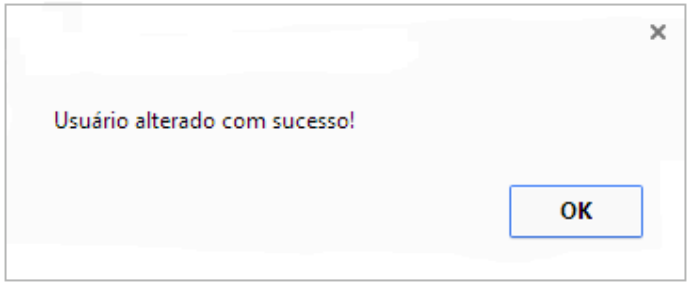


Figura 12–Confirmação de alteração

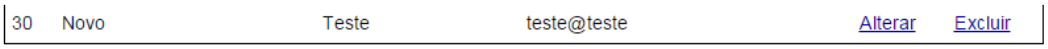


Figura 13–Listagem de usuário

Para realizar a automação do BDD e validar a mensagem de alterado com sucesso, e também validar na listagem se o usuário foi realmente alterado, iremos utilizar o seguinte código:

```
def test03AlteraUsuario
    row = $browser.table.tbody.trs.find do |tr|
      tr.td(:index => 3).text == 'teste@teste'
    end
    row.link(:text => 'Alterar').click

    #Altera os dados
    $browser.text_field(:name, 'nome').set('Alteração')
    $browser.button(:name, 'Alterar').click

    assert_equal("Usuário alterado com sucesso!", $browser.alert.text)
    $browser.alert.ok

    row = $browser.table.tbody.trs.find do |tr|
      tr.td(:index => 1).text == 'Alteração' and
      tr.td(:index => 3).text == 'teste@teste'
    end

    #valida se encontrou o email na lista
    assert_equal(6, row.cells.lenght)
  end
```

Após realizada a alteração do usuário, iremos automatizar nosso último cenário de testes, a exclusão do usuário:

Cenário 4: Excluir usuário.

Dado que estou na tela de listagem de usuários.

Quando eu clicar em excluir.

E confirmar a exclusão.

Então o sistema efetua a exclusão do usuário.

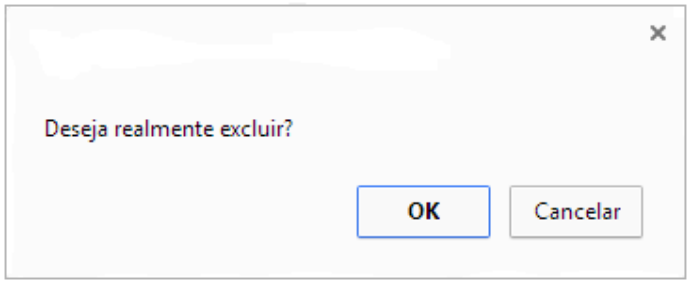
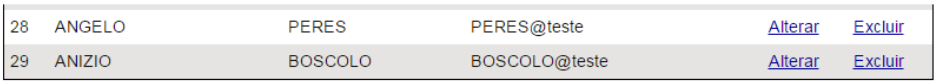


Figura 14–confirmação para exclusão de usuário



28	ANGELO	PERES	PERES@teste	Alterar	Excluir
29	ANIZIO	BOSCOLO	BOSCOLO@teste	Alterar	Excluir

Figura 14–Listagem de usuário

Para realizar a automação do BDD e validar a mensagem de alterado com sucesso, e também validar na listagem se o usuário foi realmente alterado, iremos utilizar o seguinte código:

```
def test04ExcluirUsuario
    row = $browser.table.tbody.trs.find do |tr|
      tr.td(:index => 3).text == 'teste@teste'
    end
    row.link(:text => 'Excluir').click
    assert_equal("Deseja realmente excluir?", $browser.alert.text)
    $browser.alert.ok

    #Valida se excluiu
    row = $browser.table.tbody.trs.find do |tr|
      tr.td(:index => 3).text == 'teste@teste'
    end
    assert_equal(nil, row)
 end
```

Seguindo estes passos, já temos pronta nossa primeira automação de um website. Após a execução, é possível analisar o log dos resultados obtidos durante a execução.

```
Loaded suite C:/Users/robertopegoraro/Desktop/TesteCadastroUsuarios
Started
...

Finished in 14.608325 seconds.

3 tests, 6 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
100% passed

0.21 tests/s, 0.41 assertions/s
```

Figura 15–Resultado da execução

Para poder expressar em números a vantagem da utilização da automação de testes, foi criada a seguinte tabela, expressando os valores de tempo gastos em nas etapas de execução dos cenários manualmente, e execução dos cenários automatizados para que sejá possível utilizar como base no momento de adotar ou não a estratégia de automação de testes.

Número de Execuções	**Execução dos cenários manualmente**	**Execução dos cenários automatizados**
1	75 Segundos	14 Segundos
2	62 Segundos	16 Segundos
3	68 Segundos	17 Segundos
4	60 Segundos	15 Segundos
5	70 Segundos	15 Segundos
Total	335 Segundos	77 Segundos

Tabela 4– Tempo de execução

6. CONCLUSAO

A automação de testes está sendo cada dia mais, difundida e utilizada pelas empresas de desenvolvimento de software, devido a confiabilidade e agilidade na hora de executar diversos testes.

Além da cobrança cada vez maior por parte dos clientes no que diz respeito a qualidade de software, os prazo de entrega das aplicações web estão cada vez mais curtos. Sendo assim esse trabalho de monografia visou abranger, além dos conceitos e tipos de testes descrever também a automação de testes como um todo, mostrando algumas técnicas e também a ferramenta proposta no início deste trabalho, que pode ser utilizada visando diminuir o esforço humano com atividades repetitivas e ter um retorno rápido de como está a qualidade do software.

Nesta simples automação de testes de um aplicativo web, é notório o ganho de tempo e assertividade dos testes, onde executado os testes manualmente durante um ciclo de 5 vezes, obteve um atraso de aproximadamente 75% do tempo, além da possibilidade de uma falha humana ocorrer, e passar um erro despercebido, indo este parar no cliente. Este teste é apenas um pequeno exemplo, e já foi possível perceber essa grande agilidade e confiabilidade na execução do teste. Quando se tem muitos mais casos de teste automatizados e o sistema é ainda maior e complexo, é mais notório os benefícios que a automação de testes traz.

O ponto negativo encontrado utilizando a ferramenta é que não possui uma IDE para desenvolvimento, por isso durante a criação dos primeiros scripts de testes é necessário realizar consultas na internet de como criar determinado comando, mas com o uso constante essa dificuldade diminui ou extingue-se.

REFERÊNCIAS

BARTIÉ, Alexandre. **Garantia da qualidade de software: Adquirindo maturidade organizacional**. Rio de Janeiro: Elsevier, 2002.

BECK, Kent**. Test Driven Development: By Example**. Boston. Ed, Person Education, 2003

CAETANO, Cristiano. **Engenharia de Software Magazine**, 5ª. ed., 2008

CARVALHO, Eliana L. de. **Testes de Software e Automação de Testes.** Disponível em <http://www.tmtestes.com.br/uploads/conteudo/49/Artigo_1-Teste_%20Automacao_SW.pdf>. Acesso em 05/03/2015.

ELIZA, Renata; LAGARES, Vivian. **Testes de Desempenho, Carga e Stress.** Artigo da revista Java Magazine, Ed. 110., Disponível em: <http://www.devmedia.com.br/testes-de-desempenho-carga-e-stress-revista-java-magazine-110/26546>. Acesso em: 22/09/2014.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa.** 4 ed. São Paulo: Atlas, 2008.

GONÇALVES, H.N. **Geração de Testes Automatizados utilizando o Selenium**. Trabalho de Conclusão de Universidade de Pernambuco. Escola Politécnica de Pernambuco. Graduação em Engenharia da Computação, 2011.

LOURENÇO, Marcelo. **Fases de Teste.** Disponível em <http://qualidade-de-software.blogspot.com.br/2010/02/fases-de-testes.html>. Acesso em 16/09/2014.

MALDONADO, J., C, DELAMARO, M., E., e JINO, M., **Introdução ao Teste de Software.** Ed. Elseivier, Campus, 2007.

MOREIRA, Anderson. **Testes Unitários com JUnit**. Disponível em <http://siep.ifpe.edu.br/anderson/blog/?page_id=976>. Acesso em 14/10/2014.

MYERS, Glenford J. **The Art of Software Testing.** 2ª. ed**.**, New York. Ed. John Wiley & Sons, 2004.

PETTICHORD, Disponível em <http://pettichord.com/watirtutorial/docs/watir_cheat_sheet/WTR/Methods%20supported%20by%20Element.html>. Acesso em 29/11/2014.

PEZZÈ, Mauro, Michel Young. **Teste e análise de software**. tradução Bernardo Copstein, Flavio Moreira de Oliveira. Porto Alegre: Ed. Bookman, 2008.

PIERRI, Fabio. **Fases de Testes de Software.** Disponível em <http://www.matera.com/br/2013/07/19/fases-de-testes-de-software>. Acesso em 16/09/2014.

PRESSMAN, Roger S. **Engenharia de Software**. 6ª. ed., Rio de Janeiro: Ed. McGraw-Hill, 2006.

PRESSMAN, Roger S. **Software Engineering: Practitioner's Approach**. 3ª. ed., New York: Ed. McGRAW-HILL, 1992.

RIBEIRO, Camilo, **Entendendo BDD com Cucumber – Parte I**. Disponível em <http://www.bugbang.com.br/entendendo-bdd-com-cucumber-parte-i>. Acesso em 14/10/2014

RIOS, Emerson, MOREIRA, Trayahu. **Teste de Software.** 2ª. ed., Rio de Janeiro. Ed. Altabooks. 2006.

ROCHA, Anne Caroline O. **A importância dos testes de Regressão.** Disponível em <http://gtsw.blogspot.com.br/2009/03/importancia-dos-testes-de-regressao.html>. Acesso em 09/09/2014.

RUBY, disponível em <http://rubyinstaller.org/downloads/>.

SANTOS, Ismayle de S. S.; NETO, Pedro de A. dos Santos. **Automação de testes funcionais com selenium.** Disponível em <http://www.ufpi.br/subsiteFiles/ercemapi/arquivos/files/minicurso/mc2.pdf>. Acesso em 15/03/2015

SILVA, Fernando Rodrigues da. **Testes de Software - Teste Unitário.** Disponível em <http://www.devmedia.com.br/testes-de-software-teste-unitario/22284>. Acesso em 16/09/2014.

SOARES, Ismael. **Introdução a Behavior Driven Development.**Disponível em <http://pt.slideshare.net/rkmael/introduo-a-bdd>. Acesso em 14/10/2014.

SOUSA, Markelly. **O Processo de Teste: Testes de Carga.** Disponível em <http://thetestingprocess.blogspot.com.br/2012/08/testes-de-carga.html>. Acesso em 14/10/2014.

SUBLIME, disponível em <http://www.sublimetext.com/2>.

WATIR, Disponível em <http://watir.com/>. Acesso em 25/11/2014.