



UNIVERSIDADE FEDERAL DE MATO GROSSO  
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM  
SISTEMAS DE INFORMAÇÃO

**RELATÓRIO DE ESTÁGIO SUPERVISIONADO  
TESTES AUTOMATIZADOS EM NÍVEL DE USUÁRIO  
PARA APLICAÇÕES WEB**

**JOSEFSON FRAGA SOUZA**

CUIABÁ – MT

2015

**UNIVERSIDADE FEDERAL DE MATO GROSSO**  
**COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM SISTEMAS DE**  
**INFORMAÇÃO**

**RELÁTÓRIO DE ESTÁGIO SUPERVISIONADO**  
**TESTES AUTOMATIZADOS EM NÍVEL DE USUÁRIO PARA**  
**APLICAÇÕES WEB**

**JOSEFSON FRAGA SOUZA**

Relatório apresentado no Instituto de Computação da  
Universidade Federal de Mato Grosso, para  
obtenção do título de Bacharel em Sistemas de  
Informação.

**CUIABÁ – MT**

**2015**

UNIVERSIDADE FEDERAL DE MATO GROSSO  
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM SISTEMAS DE  
INFORMAÇÃO

JOSEFSON FRAGA SOUZA

Relatório de Estágio Supervisionado apresentado à Coordenação do Curso de Sistemas de Informação como uma das exigências para obtenção do título de Bacharel em Sistemas de Informação da Universidade Federal de Mato Grosso

Aprovado por:

---

Prof. MSc. Nilton Hideki Takagi  
Instituto de Computação  
(Coordenador de Estágios)

---

Prof. MSc. Carlos Ueslei Rodrigues de Oliveira  
Instituto de Computação  
(ORIENTADOR)

---

Diego José Duarte Almeida  
(SUPERVISOR)

## **DEDICATÓRIA**

*Aos meus pais por toda educação, compreensão e apoio  
prestado não só durante a graduação, mas durante toda  
minha vida.*

## **AGRADECIMENTOS**

Agradeço primeiramente a minha família pela educação e ensinamentos morais e éticos que me possibilitaram chegar até aqui. Pelo incentivo e apoio incondicional fornecidos durante toda a minha formação acadêmica.

Aos amigos e colegas de faculdade pelo companheirismo e apoio nos estudos, e é claro por toda diversão gerada ao longo de nossos longos anos de graduação.

Aos professores do Instituto de Computação, pelo empenho em lecionar, em nos criticar positivamente nos incentivando a sempre buscar algo melhor. Em especial ao meu orientador por sua ajuda nesta fase final.

Agradeço também a empresa Braising por todo apoio para realização deste trabalho, em especial a meu supervisor por ter acreditado na proposta e apoiado este trabalho.

## SUMÁRIO

|   |           |
|---|-----------|
| <b>LISTA DE FIGURAS.....</b>                                    | <b>8</b>  |
| <b>RESUMO.....</b>  | <b>9</b>  |
| <b>INTRODUÇÃO.....</b>  | <b>10</b> |
| <b>1. REVISÃO DE LITERATURA.....</b>                            | <b>11</b> |
| 1.1 TESTE DE SOFTWARE.....                                      | 11        |
| 1.2 ATIVIDADES DE TESTES.....                                   | 12        |
| 1.3 AUTOMAÇÃO DE TESTES.....                                    | 13        |
| 1.3.1 <i>Promessas E Problemas De Testes Automatizados.....</i> | <i>13</i> |
| 1.3.2 <i>Diferentes Métodos De Testes De Automação.....</i>     | <i>14</i> |
| 1.4 FRAMEWORK.....  | 17        |
| 1.5 FRAMEWORKS DE AUTOMAÇÃO DE TESTES.....                      | 17        |
| 1.6 REQUISITOS DE UM FRAMEWORK DE AUTOMAÇÃO DE TESTES.....      | 18        |
| <b>2. MATERIAIS, TÉCNICAS E MÉTODOS.....</b>                    | <b>20</b> |
| 2.1 FERRAMENTAS UTILIZADAS.....                                 | 20        |
| 2.1.1 <i>Python Como Linguagem.....</i>                         | <i>21</i> |
| 2.1.2 <i>Selenium.....</i>                                      | <i>22</i> |
| 2.1.3 <i>MozilLa Firefox.....</i>                               | <i>23</i> |
| 2.1.3.1 <i>Firebug Addon.....</i>                               | <i>23</i> |
| 2.1.3.2 <i>Firepath Addon.....</i>                              | <i>23</i> |
| 2.1.4 <i>Git.....</i>   | <i>24</i> |
| 2.1.5 <i>Jenkins.....</i>                                       | <i>24</i> |
| 2.2 PAGE OBJECT DESIGN PATTERN.....                             | 25        |
| <b>3. RESULTADOS.....</b>                                       | <b>27</b> |
| 3.1 CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO.....            | 27        |
| 3.2 CRIAÇÃO DAS CLASSES BASE.....                               | 30        |
| 3.3 ENTENDENDO O PAGE OBJECT PATTERN.....                       | 33        |
| 3.4 ESCRREVENDO UMA PÁGINA.....                                 | 34        |
| 3.5 ESCRREVENDO UM TEST CASE.....                               | 39        |
| 3.6 SELENIUM GRID.....  | 41        |

|   |           |
|---|-----------|
| 3.7 JENKINS.....                                | 45        |
| <b>4. DIFICULDADES ENCONTRADAS.....</b>         | <b>53</b> |
| <b>5. CONCLUSÕES.....</b>                       | <b>54</b> |
| <b>6. LISTA DE SIGLAS E ABREVIATURAS.....</b>   | <b>55</b> |
| <b>7. REFERÊNCIAS BIBLIOGRÁFICAS.....</b>       | <b>57</b> |
| <b>ANEXO I – QUESTIONÁRIO – PERGUNTAS.....</b>  | <b>59</b> |
| <b>ANEXO II – QUESTIONÁRIO – RESPOSTAS.....</b> | <b>60</b> |

## LISTA DE FIGURAS

|   |    |
|---|----|
| Figura 1: Page object design.....                             | 26 |
| Figura 2: Virtualenv – Instalação e criação.....              | 28 |
| Figura 3: Virtualenv – Dependências.....                      | 28 |
| Figura 4: Esqueleto framework.....                            | 29 |
| Figura 5: Framework – Páginas e Test Suites.....              | 30 |
| FFigura 6: Página base.....                                   | 31 |
| Figura 7: Classe base de test cases.....                      | 32 |
| Figura 8: Página de login.....                                | 34 |
| Figura 9: Localizando um Identificador.....                   | 35 |
| Figura 10: Página LoginRegister 1.....                        | 37 |
| Figura 11: Página LoginRegister 2.....                        | 38 |
| Figura 12: LoginTest – Test Cases.....                        | 40 |
| Figura 13: Selenium grid – Schema.....                        | 42 |
| Figura 14: Selenium grid – Hub.....                           | 43 |
| Figura 15: Selenium grid – Configuração de nó.....            | 43 |
| Figura 16: Selenium grid – Nó.....                            | 44 |
| Figura 17: Selenium grid – Registro de nó.....                | 44 |
| Figura 18: Selenium grid – Recursos.....                      | 45 |
| Figura 19: Jenkins – Página de configuração.....              | 46 |
| Figura 20: Jenkins – Configurações Gerais.....                | 46 |
| Figura 21: Jenkins – Nova tarefa.....                         | 47 |
| Figura 22: Configurando tarefa – SMC.....                     | 47 |
| Figura 23: Configurando tarefa – Gatilho.....                 | 48 |
| Figura 24: Configurando tarefa – Notificação após commit..... | 48 |
| Figura 25: Notificação pós commit.....                        | 48 |
| Figura 26: Configurando tarefa - Build.....                   | 49 |
| Figura 27: Configurando tarefa - Ação pós build.....          | 49 |
| Figura 28: Build - Resumo.....                                | 50 |
| Figura 29: Build - Commit responsável.....                    | 50 |
| Figura 30: Build - Resultado de testes.....                   | 51 |
| Figura 31: Build -Test Suite – Test Cases.....                | 51 |
| Figura 32: Build – Falha em test case.....                    | 52 |



## RESUMO

O natureza deste trabalho é apresentar as atividades desenvolvidas no estágio supervisionado junto a empresa Braising no período de abril à junho de 2015.

Durante este período a principal função deste trabalho foi desenvolver um framework de testes automatizados em nível de usuário para uma das aplicações web desenvolvidas e mantidas pela empresa em questão. Para tal, foram utilizadas ferramentas como a biblioteca Selenium, a ferramenta de *continuous integration* (CI) Jenkins, um framework de testes da família xUnit e *design patterns*. Demais tecnologias, técnicas e métodos serão listados na devida seção deste documento.

O resultado é a entrega de um framework de testes automatizados resilientes a mudanças na aplicação, com baixíssimo custo de manutenção, com enorme potencial de reuso e expansão de suas funcionalidades.

## INTRODUÇÃO

O presente relatório trata das atividades desenvolvidas no estágio supervisionado, com objetivo de desenvolver um framework de testes automatizados para uma aplicação web mantida pela empresa Braising.

A Braising é uma *software house* de pequeno porte focada principalmente no desenvolvimento de aplicações web. Como muitas outras empresas do ramo, a Braising utiliza técnicas e metodologias de desenvolvimento ágil para alcançar maior produtividade.

A necessidade de um framework de testes automatizados em nível de usuário se apresenta não só na Braising, onde cada desenvolvedor testa o que produz, mas também no mercado de *software houses* em geral, principalmente em empresas de pequeno e médio porte onde o desenvolvimento ágil é largamente praticado.

Tal necessidade pode ser notada dada uma pesquisa realizada pelo autor deste trabalho na cidade de Cuiabá, através do *google forms*, e distribuída através de listas de e-mails a profissionais de desenvolvimento de *software houses* na cidade.

Nesta pesquisa, 95.5% dos entrevistados alegaram que a empresa testa o software por ela produzido, contudo só 72.7% realizam testes em nível de usuário. E apesar de 81.8% afirmarem que testes são executados a cada versão ou iteração do processo de desenvolvimento, apenas 27.3% o fazem de alguma maneira automatizada.

Este trabalho, vem através desse contexto, montar uma estrutura de testes automatizados para uma aplicação web escolhida junto a empresa que se enquadra como a maioria relatada na pesquisa realizada e não possui automações de teste em nível de usuário.

Ao longo deste documento serão detalhadas as atividades realizadas no período em questão, bem como as técnicas, métodos e ferramentas utilizadas para alcançar o objetivo final.

O trabalho se inicia descrevendo o cenário e motivos que levaram realização deste projeto. Logo depois, é feita uma revisão da literatura sobre assuntos envolvidos nas seções posteriores, tentando desta forma pavimentar o caminho para o decorrer deste documento. Em seguida são descritos os materiais, técnicas e métodos utilizados ao longo das atividades realizadas. Logo após são apresentados os resultados, onde serão mostradas e descritas as funcionalidades conseguidas através do produto final. Depois, as dificuldades encontradas são mostradas e finaliza-se com as conclusões.

## 1. REVISÃO DE LITERATURA

Durante as atividades do estágio vários conceitos trabalhados dentro e fora de aula foram de fundamental importância. Neste capítulo serão apresentados os conceitos que serviram como base para a execução das atividades descritas neste relatório.

### 1.1 TESTE DE SOFTWARE

Define-se teste de software o processo de operar um sistema ou componente sobre condições específicas, observando ou gravando os resultados e fazendo uma avaliação de algum aspecto do sistema ou componente (“IEEE Standard Glossary of Software Engineering Terminology”, 1990).

#### **Smoke Tests**

Também conhecido como *sanity tests* ou *confidence tests*, *smoke tests* são aqueles testes cuja única intenção é demonstrar que a funcionalidade mais básica do sistema está funcionando. São frequentemente utilizados como testes preliminares, os quais respondem perguntas muito básicas do sistema: O sistema está rodando?; A janela principal está aberta?; Os menus estão respondendo?; etc.

#### **Testes De Regressão**

Testes de regressão é a ação de re-testar componentes ou sistemas previamente testados para assegurar que defeitos não tenham sido gerados ou descobertos a partir de mudanças recentes. São amplamente utilizados em caso de mudanças no software ou no ambiente (CRAIG; JASKIEL, 2002).

## Testes De Aceitação

Um nível de testes conduzidos da visão do cliente ou usuário, usados para estabelecer critérios de aceitação para um sistema. Geralmente baseados nos requisitos do sistema (CRAIG; JASKIEL, 2002).

## 1.2 ATIVIDADES DE TESTES

Teste de software é muito mais do que simplesmente executar *test cases*, da mesma forma, automação de testes não é limitada somente a automação da execução dos testes. Nesta seção, são decompostas as etapas para criação de uma ferramenta automatizada de testes.

### Projetando Os Test Cases

Sistemas funcionais e testes de aceitação são projetados por engenheiros de testes utilizando os requisitos do sistema e técnicas de design de testes. Construir bons *test cases* não é uma tarefa fácil e é uma das principais habilidades que um profissional de engenharia de testes deve possuir (CRAIG; JASKIEL, 2002; FEWSTER; GRAHAM, 1999).

### Executando Os Test Cases E Analisando Os Resultados

Depois dos *test cases* terem sido projetados e criados, eles podem ser executados e ter seus resultados verificados. Estes testes são frequentemente re-executados quando uma nova versão do sistema sobre teste é disponível.

Automatizar estes testes de regressão, ou pelo menos um subconjunto deles, frequentemente chamados de *smoke tests*, torna a execução de testes e a análise de resultados muito mais rápidas.

### Feedback Dos Resultados

Depois dos engenheiros de testes terem rodado seus testes, eles devem relatar os resultados para os responsáveis cabíveis, como por exemplo o gerente do projeto. Se a

automação é totalmente automatizada, faz sentido automatizar também o *feedback* nesta fase.

### 1.3 AUTOMAÇÃO DE TESTES

Dá-se o nome de automação de testes ao uso de software para controlar a execução de testes, a comparação entre os resultados do teste e os resultados previstos, a configuração de condições pré-existent e outras funções de controle e o *feedback* dos resultados dos testes (“BS 7925-1”, 2005).

#### 1.3.1 PROMESSAS E PROBLEMAS DE TESTES AUTOMATIZADOS

A automação pode facilitar a carga de trabalho de engenheiros de testes e liberá-los de tarefas repetidas. Isso apresenta o potencial de aumentar a qualidade do software e diminuir o tempo de teste.

Ser capaz de executar testes previamente criados sem nenhum trabalho extra sobre novas versões de sistema claramente torna a fase de testes mais eficiente. A automação, além de possibilitar a execução de testes de forma mais rápida, o que significa mais e mais rodadas de testes, também torna a tarefa de criar novos test cases mais fácil e mais rápida. Ou seja, automatizar nos permite rodar mais testes com maior frequência. Tudo isso, inerentemente, provê uma melhor utilização de recursos.

Além disso, a reutilização de testes e a diminuição do tempo de execução dos mesmos acelera o ciclo de *feedback* para os desenvolvedores. Isso, somado ao aumento de confiança que toda a equipe ganha por ter uma rotina de testes automatizados rodando frequentemente de forma consistente em diferentes ambientes, acaba por reduzir o tempo que o produto leva para chegar até o mercado.

Contudo, apesar da automação de testes proporcionar inúmeras vantagens, ela não é uma tarefa fácil, apresenta muitos problemas e tem que ser executada com muita cautela.

Problemas como expectativas irreais, onde gerentes acreditam que a automação resolverá todos os seus problemas relacionados a testes e magicamente fará o software ter mais qualidade. Expectativas que testes automatizados acharão muitos novos defeitos de software. Falso senso de segurança, onde só porque a aplicação não falhou nos testes, não significa que a aplicação esteja 100% livre de erros.

Na realidade, o processo de automatização encontra mais erros durante o

desenvolvimento do framework do que quando os testes são rodados. Uma vez que os testes estiverem rodando, eles só devem quebrar quando houver mudanças nos requisitos funcionais que acarretem alterações de interface ou de ações de usuários.

Essa característica implica na capacidade de manutenção dos testes, pois uma vez que o sistema muda, os testes também mudam. Engenheiros de testes testando um sistema manualmente são capazes de se adaptarem a grandes mudanças sem problemas, já os testes automatizados podem falhar mesmo nas menores mudanças. E se o trabalho da manutenção nos testes automatizados levar mais tempo que testar manualmente, a prática de testes automatizados certamente será abandonada. O mesmo acontecerá se a tarefa de adicionar novos recursos for muito complicada.

O problema com testes automatizados parece estar na falta de percepção que qualquer projeto de automação de testes pode se tornar grande o suficiente para ser tratado como um projeto de software por si mesmo. Projetos de software falham se não forem tratados devidamente, e testes automatizados não são diferentes.

Apesar de que todas as promessas fazem a automação de testes parecer muito atrativa, alcançar tal automação na realidade exige muito trabalho. De forma que, se a automação não for bem feita, ela será rapidamente abandonada e as promessas não serão alcançadas (FEWSTER; GRAHAM, 1999).

### **1.3.2 DIFERENTES MÉTODOS DE TESTES DE AUTOMAÇÃO**

Testes automatizados podem ser divididos em várias categorias. Esta seção introduz brevemente as principais categorias nas quais testes automatizados podem ser divididos.

#### **Testes Dinâmicos E Testes Estáticos**

Basicamente, enquanto em testes dinâmicos algo é executado no sistema em teste e o status do teste é verificado ao seu término, num teste estático a aplicação sobre teste não é executada de forma alguma.

Exemplos de testes estáticos são revisões e análise estática de código (Syntax e complexidade do código). Enquanto revisões são feitas principalmente por humanos, análises estáticas são geralmente executadas por computadores.

## Testes Funcionais E Não-funcionais

Testes dinâmicos ainda podem ser divididos em testes funcionais e testes não-funcionais. O primeiro, como o próprio nome indica, vem assegurar que a funcionalidade do sistema se adere aos requisitos. Já o último, serve para aferir que os outros aspectos, não-funcionais do sistema sobre teste também funcionem (“IEEE Standard Glossary of Software Engineering Terminology”, 1990).

Há muito o que testar além da funcionalidade. Testar áreas como performance, segurança, usabilidade, portabilidade e confiança é muito importante e falhas em qualquer destes setores podem arruinar um produto ou até mesmo uma empresa. No entanto, apesar da importância dos testes não-funcionais, o escopo deste trabalho estará focado primordialmente nos testes funcionais.

## Granularidade

Os testes ainda podem ser divididos em diferentes níveis, como: testes de unidade, testes de integração, testes de sistema e testes de aceitação (BURNSTEIN, 2003; CRAIG; JASKIEL, 2002).

## Unit Testing

O menor bloco na construção de um sistema, a isso chamamos de unidade. Unidades geralmente possuem uma *application programming interface* (API) que são utilizadas quando interagem com outras unidades e que também pode ser utilizada para testá-las. Na maioria dos casos *unit tests* são gerenciados pelos próprios desenvolvedores, os quais conhecem o código sobre teste (CRAIG; JASKIEL, 2002).

*Unit tests* são automatizados por natureza e utilizam em sua grande maioria os xUnit frameworks, como o JUnit para o java, PyUnit para o python e o NUnit para o .NET.

*Unit tests* podem ainda ser levados a outro nível quando usados em conjunto com *test driven development* (TDD). Neste cenário os testes são criados antes mesmo do código de produção e como resultado, o design é limpo e o número de erros é baixo, visto que tal metodologia força os desenvolvedores a pensar sobre o design das unidades.

O escopo deste trabalho está em testes automatizados de alto nível, sob a perspectiva do cliente ou usuário do sistema. O que não significa que testes unitários e sua automação sejam menos importante. Pelo contrário, os primeiros esforços na automação de testes deveriam ser empregados diretamente no nível de testes unitários, onde os investimentos são pequenos e o retorno é rápido.

### **Teste De Componentes**

Componentes variam muito de sistema a sistema, mas de forma generalizada um componente é uma coleção de unidades relacionadas que tem uma interface em comum em relação a outros componentes.

Automação neste nível não é muito difícil. Ferramentas de testes podem ser ligadas a mesma interface que é utilizada por outros componentes, desta forma pode-se utilizar ferramentas que dirigem a interface utilizada pelo componente a ser testado.

### **Teste De Sistema**

A diferença entre um componente e um sistema é que sistemas funcionam por si só enquanto componentes só são utilizados como parte de um sistema. Mas na realidade esta diferença pode parecer vaga quando frequentemente sistemas são integrados uns a outros de forma a formar um sistema ainda maior.

Sistemas possuem diferentes interfaces em relação ao mundo. Dependendo de quem estiver a utilizá-lo, seja um humano ou outros sistemas. Em relação a interfaces, sistemas utilizados por humanos possuem algum tipo de interface de usuário, podendo esta ser gráfica ou não-gráfica, enquanto sistemas usados por outros sistemas possuem interfaces similares na forma de um componente. É importante notar que é muito mais difícil automatizar interfaces gráficas de usuário.

### **Black-Box Testing**

Este é um tipo de teste onde o funcionamento do sistema é desconhecido e ignorado. Onde os testes são realizados para verificar se o sistema faz o que é suposto. Tipicamente



utilizado para realização de testes funcionais, onde o principal objetivo é validar os requisitos funcionais do sistema (CRAIG; JASKIEL, 2002).

Este trabalho utilizará esta categoria de teste para validar as principais funcionalidades do sistema. As quais serão testadas frequentemente a cada modificação do software. Desta forma, será possível criar testes de regressão automatizados testando as principais funcionalidades do sistema em nível de usuário, os quais também podem ser utilizados como testes de aceitação.

## **1.4 FRAMEWORK**

Uma das mais importantes formas de reuso é o reuso de design. Utilizando-se de coleções de classes abstratas é possível expressar um design abstrato. Design este que pode ser estendido, adicionando mais ou melhores componentes a ele (JOHNSON; FOOT, 1988).

Frameworks são reconhecidos principalmente por sua capacidade de reuso de código e por sua extensibilidade. Apesar deles serem semelhantes a bibliotecas, eles apresentam diferenças bem definidas e ambos não devem ser confundidos.

Uma característica importante de um framework é que o diferencia de bibliotecas é que os métodos definidos na construção do framework são chamados de dentro do próprio framework. Desta forma, frameworks têm um comportamento padrão, visto que o fluxo de controle é controlado pelo próprio framework e não por invocações de usuário (JOHNSON; FOOT, 1988).

## **1.5 FRAMEWORKS DE AUTOMAÇÃO DE TESTES**

Quando a automação de mais alto nível é alcançada, testes devem ser executados com um simples apertar de botão ou automaticamente. Este tipo de automação requer algum tipo de sistema, o qual possibilita fazer da criação, execução e manutenção dos testes uma tarefa fácil. O sistema deve prover algumas funcionalidades-chaves bem como a extensão do mesmo, permitindo a fácil inclusão de novos testes e até novas funcionalidades. Tal software, se alinha com a definição de framework de Johnson (JOHNSON; FOOT, 1988). Desta forma, é apropriado categorizarmos este tipo de sistema como um framework de testes automatizados. Tais frameworks evoluíram ao longo do tempo e são hoje classificados em três gerações (KIT, 1999).

## Frameworks De Primeira Geração

Os frameworks de primeira geração não eram estruturados, eles possuíam dados de teste junto aos scripts e normalmente existia um script por test case. Estes *scripts* eram, em sua grande maioria, gerados com a utilização de ferramentas de *record and play*, apesar de poderem ser codificados manualmente.

Estes *scripts* apresentavam um custo inicial muito baixo, contudo o custo de manutenção agregado a eles é altíssimo, fazendo ser necessário, sempre que houver alguma mudança do sistema, a re-gravação ou a re-escrita dos *scripts*.

## Frameworks De Segunda Geração

Os frameworks de segunda geração já eram melhor projetados, tendo como principais atributos modularidade e robustez, os quais traziam melhor capacidade de manutenção. Estes scripts não só lidavam com a execução dos testes mas também pré-configurações, limpezas pós testes, detecções de erro e recuperação sobre os mesmos. Nesta geração, os *scripts* eram em sua grande maioria, escritos manualmente e tanto sua implementação quanto sua manutenção exigiam habilidades de programação, as quais nem sempre engenheiros de teste possuíam.

## Frameworks De Terceira Geração

Esta geração de frameworks além de herdar todas as boas características da sua geração antecessora, apresentam dois novos benefícios. O primeiro é que um *driver script* pode executar vários *test cases* similares e adicionar novos testes de forma trivial. E o segundo é que o design dos testes e a implementação do framework são duas tarefas diferentes. O framework desenvolvido neste trabalho foi desenhado para atender as características dessa geração de frameworks.

## 1.6 REQUISITOS DE UM FRAMEWORK DE AUTOMAÇÃO DE TESTES

A execução automática dos testes é, sem dúvida o requisito número um para um

framework de automação de testes. Contudo, só a execução dos testes não é suficiente, o framework também deve ser capaz de analisar os resultados dos testes e reportar os resultados (KIT, 1999).

O framework deve ser capaz de iniciar a execução dos testes com um simples apertar de botão ou de forma automática. Isto significa que o framework deve ser capaz de construir o ambiente de testes e se possível assegurar que as pré-condições foram alcançadas (FEWSTER; GRAHAM, 1999).

O framework deve então verificar os resultados dos testes comparando estes resultados aos resultados esperados e atribuir um status a esta resposta. Caso os testes tenham sido executados sem problemas e todas as comparações forem verdadeiras o teste recebe o status de sucesso, caso contrário o status de falha. Além disso, deve-se prover logs detalhados para que em casos de falhas, o trabalho do entendimento do erro e sua correção sejam facilitados (FEWSTER; GRAHAM, 1999).

## 2. MATERIAIS, TÉCNICAS E MÉTODOS

Para o desenvolvimento das atividades realizadas durante o estágio diversos recursos tecnológicos e metodológicos foram utilizados. Nesta seção detalharemos as principais ferramentas e técnicas utilizadas de forma a justificar sua utilização.

### 2.1 FERRAMENTAS UTILIZADAS

Além de respeitar ao máximo aos requisitos necessários para criação do framework de testes, foram considerados fatores que facilitassem a adoção destas ferramentas, principalmente em ambientes de desenvolvimento ágil.

Nagy afirma que softwares *open source* têm penetrado cada vez mais no mundo comercial, principalmente na última década. De forma que estas aplicações desenvolvidas por grupos de desenvolvedores voluntários tem o potencial necessário para quebrar a dominância de softwares proprietários para muitas aplicações de negócio. Ele atribui tal avanço no mercado a fatores como a flexibilidade de customização para diferentes necessidades, a frequente utilização de padrões abertos, e a economia em licenças no lado gerencial. Empresas como a Amazon foram citadas por cortar o gasto em tecnologias em quase um terço, meramente pela adoção de softwares *open source* (NAGY; YASSIN; BHATTACHERJEE, 2010).

Já Wang faz um estudo na adoção de softwares *open source* onde afirma que a escolha do software correto é crítica para o sucesso do projeto. Além disso ressalta que, desde que os softwares *open source* entraram no mercado as opções de escolha ficam cada vez mais abrangentes e mais confusas (WANG; WANG, 2001).

Dessa forma, visando a melhor adoção desse framework na empresa o autor prezou pela escolha de softwares que trouxessem os benefícios listados abaixo.

#### **Suporte**

Nenhum software é livre de defeitos, mais cedo ou mais tarde precisaremos de algum tipo de ajuda para resolvermos algum problema que encontramos com o software utilizado.

Aqui é importante sabermos que tipo e a qualidade do suporte provido pela empresa ou comunidade relacionada ao software.

Todos os softwares abaixo possuem grandes comunidades e suporte sobre ela em diferentes canais de comunicações, como listas de e-mail e canais irc. Neste último, centenas de pessoas respondem perguntas e resolvem problemas alheios em tempo real.

## **Custo**

Preço geralmente é um limitador na aquisição de novos softwares. Contudo, todas as ferramentas aqui listadas são livres de custo de licenças por serem *open source*.

## **Compatibilidade**

O software é compatível com os ambientes operacionais existentes na empresa? As ferramentas aqui listadas, são compatíveis com todos os grandes sistemas operacionais existentes, Windows, Linux e seus derivados e OSX.

### **2.1.1 PYTHON COMO LINGUAGEM**

Uma linguagem se faz necessária na construção de testes automatizados e a disponibilidade de múltiplas linguagens pode fazer a escolha da linguagem correta se tornar uma tarefa difícil. Geralmente escolher uma linguagem já utilizada na organização é uma aposta segura, mas este não deve ser o único critério.

Observou-se na referência literária que para o sucesso de uma solução de testes automatizados são necessários alguns requisitos, estando entre os mais importantes a capacidade de manutenção e a facilidade de uso. Pois, caso estes dois requisitos não sejam atendidos, o projeto provavelmente será abandonado por seus usuários.

Sendo assim, a fim de buscar uma linguagem que promova tais requisitos, uma vez que, na maioria das vezes os responsáveis pela execução e manutenção dos testes não são programadores, escolheu-se neste trabalho, python como linguagem para o projeto.

Python é uma linguagem de propósito geral de alto nível e sintaxe muito clara, cuja filosofia de design tem como ênfase a legibilidade do código, o que por si só reduz o custo de

manutenção do mesmo. A legibilidade de código está tão embrenhada no coração da linguagem que python é a única linguagem de grande porte que usa indentação como definição de blocos de código. Certamente, um alto nível de legibilidade está no coração do design da linguagem, que apenas reconhece o fato que qualquer código é muito mais lido do que escrito (RASHED; AHSAN; OTHERS, 2012; ROSSUM; DRAKE, 2002).

Outro ponto importante é a economia de tempo trazida com a produtividade embutida em linguagens dinamicamente interpretadas. Onde, diferentemente de linguagens de sistemas, como linguagens da família C, ela não requer processos de compilação, o que acaba por acelerar o desenvolvimento. Além disso, (OUSTERHOUT, 1998) sugere que programar algo numa linguagem de script utiliza-se de 5 a 10 vezes menos código que numa linguagem de sistema.

Alguns podem levantar a questão da performance, onde as linguagens de script são muito mais lentas que linguagens de sistemas, contudo os computadores têm se tornado cada vez mais velozes ao ponto que isto já não é tão importante. Além disso, esta diferença de performance em automação de testes não é de grande importância pois, de qualquer forma, a maioria do tempo gasto na execução de testes é esperando o sistema responderem às entradas dadas.

De uma forma mais simples, ao escolher uma linguagem interpretada sobre uma linguagem de sistema, trocamos a velocidade de execução por alta produtividade de programação e capacidade de reuso. Esta troca faz sentido sob a perspectiva que cada vez os computadores estão mais rápidos e que o gargalo destes testes é a espera de sistema e não a execução do software em si.

### 2.1.2 SELENIUM

Selenium é uma coleção de ferramentas de automação de browsers *open source* criada para auxiliar na tarefa de automatização de testes em aplicações web, entretanto não é somente limitada a isso.

Um dos atributos chaves do Selenium é a habilidade de poder executar testes em diversos browsers, em diversas versões e nos três principais sistemas operacionais (Windows, Linux e Mac). Além disso, Selenium suporta uma grande variedade de linguagens de programação como Java, C#, Ruby, Perl, Python e outras (GOJARE; JOSHI; GAIGAWARE, 2015).

Para a realização deste trabalho foram utilizados o webdriver juntamente com o Selenium Grid. O Webdriver nos fornece uma API de comunicação com uma abstração de browser. Desta forma, através do Webdriver podemos fazer introspeções nas páginas web e executar ações em seus elementos. Já o Selenium Grid foi utilizado de forma a oferecer uma rede de recursos, sobre os quais os testes podem ser rodados. Estes recursos são a combinação de diversos requisitos, mas neste trabalho utilizamos apenas os seguintes:

- Sistema operacional
- Browser
- Versão do browser

### 2.1.3 MOZILLA FIREFOX

O Firefox é um navegador *open source* disponibilizado pela Mozilla, e sempre foi um dos navegadores favoritos utilizados pelos desenvolvedores, sendo a principal razão a sua vasta coleção de *addons* e seu rico modo de inspeção (MITCHELL, 2012). Por estes motivos este trabalho utilizará este browser de forma a auxiliar as tarefas desenvolvidas.

Visto que o código a ser criado é altamente dependente de identificadores de elementos html, é preciso um bom navegador para identificar tais elementos e extrair tais identificadores da melhor maneira possível.

#### 2.1.3.1 FIREBUG ADDON

Firebug é um *addon open source* que se integra ao navegador Firefox de forma a criar um, ainda mais rico, ambiente de desenvolvimento. Ele estende o Firefox de forma a prover edição, *debug* e monitoramento de elementos css, html e javascript em tempo real em qualquer página (MITCHELL, 2012). Este trabalho utilizará o Firebug principalmente para inspecionar elementos html e explorar o DOM.

#### 2.1.3.2 FIREPATH ADDON

Firepath é um *addon open source* também utilizado no Firefox que se integra ao Firebug. Sua excelência é na extração dos identificadores XPath e seletores CSS, os quais

serão utilizados para identificação de elementos quando os mesmos não possuírem melhores identificadores únicos (THOLENCE, 2014).

#### 2.1.4 GIT

Nenhuma pessoa precavida inicia algum projeto sem alguma estratégia de backup. Para projetos envolvendo textos e códigos esta estratégia de backup geralmente envolve um controle de versão (LOELIGER; MCCULLOUGH, 2012).

Git é um sistema de controle de versões distribuído e *open source* projetado para gerenciar desde pequenos a grandes projetos. Criado inicialmente por Linus Trovalds para gerenciar o código fonte do sistema operacional Linux, Git teve como prioridade sua velocidade e ser distribuído (TROVALDS, 2015).

Sua popularidade têm crescido significativamente ao longo dos anos desde sua criação. E sua adoção é notável não só na comunidade *open source* mas também em ambientes corporativos onde o termo descentralizado é um termo chave (MCKAY, 2014).

Este trabalho estará utilizando o Git para gerenciar tanto o código da aplicação web sobre teste quanto do framework de testes.

#### 2.1.5 JENKINS

Jenkins é um sistema de integração contínua *open source* o qual provê serviços para o processo de desenvolvimento de software. No mundo de desenvolvimento ágil e distribuído no qual vivemos, têm sido cada vez mais difícil saber a origem de uma quebra de sistema ou componente.

Somos incentivados a utilizarmos pequenas incrementações de *commits*, uma vez que trabalhando distributivamente, esta é a melhor forma de evitar o *merge hell*. Sendo assim com *commits* cada vez mais frequentes e de forma distribuída, como podemos saber a origem de uma quebra de sistema ou componente? O Jenkins vem atacar este problema oferecendo serviços de builds automáticas a cada *commit*, monitorando um diretório que utiliza algum sistema de controle de versão (FOWLER, 2015).

De forma análoga a uma build, podemos também vincular baterias de testes de sistemas ou componentes após algum *commit*. Desta forma mesmo que uma build não esteja quebrada, podemos ainda aferir erros na aplicação. E é desta forma que utilizaremos o Jenkins



como um CIS neste trabalho, configurando-o para monitorar um VCS de forma a rodar testes automatizados a cada novo *commit*. Guardando os relatórios gerados e alertando os responsáveis em casos de erros a serem reparados.

## 2.2 PAGE OBJECT DESIGN PATTERN

De forma a preservarmos um dos requisitos mais básicos e tão necessário em um ambiente de testes utilizaremos uma *pattern* cujo principal objetivo é a capacidade de manutenção do código produzido. Vimos que a primeira geração de frameworks de testes automatizados apresentava um grande defeito quanto a dificuldade de manutenção do mesmo.

Sendo assim, atacaremos este problema de forma a separarmos logicamente a aplicação dos testes. Desta forma, reduzindo a repetição de código pela aplicação preservando o princípio *DRY* (don't repeat yourself) da engenharia de software. O que naturalmente aumenta a capacidade de manutenção de qualquer software e aumenta o potencial de re-uso do mesmo (WILSON et al., 2014).

O *page object pattern* é um dos mais utilizados entre os testadores de aplicações web. Utilizado para estruturar os testes, de forma a separá-los das ações de baixo nível, as quais lidam com elementos web, provendo desta forma uma abstração de alto nível muito mais próxima do negócio.

Assim, esse padrão nos aconselha a criar uma classe para cada “página” da aplicação sobre teste. Definindo uma classe para cada página de forma a modelar os atributos e ações pertinentes a tal página. Isto cria uma camada de separação entre o código de teste e a implementação técnica das páginas. Como resultado, os objetos de páginas oferecerão uma API de alto nível para os testes lidarem com a funcionalidade das páginas, como pode ser verificado na Figura 1 (FOWLER, 2013).

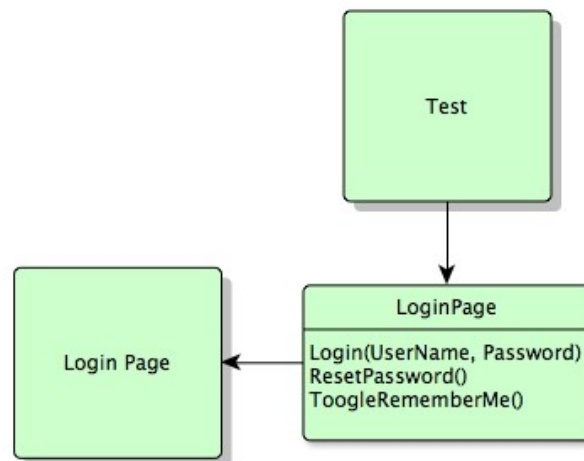


Figura 1: Page object design.

É importante notar que o conceito de página aqui é um tanto diferente. Devemos extrair grupos de elementos html no mapa do site que caracterizam uma página como única. Por exemplo, um campo de busca espalhado em toda página do site é uma página por si só, assim como um *header*, um *footer* ou um *dashboard* espalhado por páginas administrativas. Desta forma, preservamos o princípio *DRY* e aumentamos a capacidade de manutenção do nosso produto.

### 3. RESULTADOS

Nesta seção serão apresentadas as atividades que foram realizadas ao longo do período de estágio de forma detalhada. As atividades foram desenvolvidas buscando a criação de um framework de testes automatizado para uma aplicação web específica, de forma a facilitar e assegurar melhor qualidade no workflow de trabalho na empresa.

Para alcançar o resultado deste trabalho foram necessárias a realização de várias tarefas, as quais quando somadas acabam por construir a nossa solução automatizada de testes. Segmentaremos estas tarefas em uma lista, a fim de um melhor entendimento de todo o processo:

- Desenvolvimento do framework
  - Ambiente de desenvolvimento
  - Abstrações de páginas
  - *Test cases*
- Instalação e configuração do Selenium Grid
- Instalação e configuração do Jenkins

O framework nada mais é que a separação entre os *test cases* e abstrações das páginas da aplicação. Utilizando o *page object desing*, diversas páginas foram criadas de forma a oferecer uma API de alto nível composta por métodos que representam ações de usuários na página. Esta API desenvolvida será posteriormente consumida pelos *test cases* a fim de realizar e aferir ações nas páginas da aplicação.

Com esta separação, nosso framework encontra a resiliência proposta, uma vez que qualquer mudanças na funcionalidade das aplicação que venha a quebrar os testes, é de fácil manutenção. Tal manutenção se dá no framework, especificamente nas páginas que tiveram sua funcionalidade alteradas. A atualização da abstração das páginas em questão fará todos os testes que foram quebrados voltarem a funcionar.

#### 3.1 CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Criou-se um ambiente virtual com auxílio de algumas bibliotecas do repositório python a fim de encapsular o projeto, separando-o do resto do sistema. Uma vez que novas bibliotecas poderiam vir a ser utilizadas no decorrer do projeto é ideal termos um container de

isolamento entre a instalação e configuração do python e bibliotecas utilizadas no nosso projeto e do sistema.

Para isso foi utilizado o virtualenv e virtualenvwrapper, dois pacotes python que nos ajuda no gerenciamento desse ambiente virtual. Os quais podem ser instalados a partir do gerenciador de pacotes python, pip. Sendo assim, criou-se um ambiente virtual com o comando mkvirtualenv como mostrado abaixo na Figura 2.

```
josefson@padawan ~$ which python [8:36:05]
/usr/local/bin/python
josefson@padawan ~$ pip install virtualenv virtualenvwrapper [8:36:14]
Collecting virtualenv
  Using cached virtualenv-13.1.2-py2.py3-none-any.whl
Collecting virtualenvwrapper
  Using cached virtualenvwrapper-4.7.1-py2.py3-none-any.whl
Requirement already satisfied (use --upgrade to upgrade): virtualenv-clone in /usr/local/lib/python2.7/site-packages (from virtualenvwrapper)
Requirement already satisfied (use --upgrade to upgrade): stevedore in /usr/local/lib/python2.7/site-packages (from virtualenvwrapper)
Installing collected packages: virtualenv, virtualenvwrapper
Successfully installed virtualenv-13.1.2 virtualenvwrapper-4.7.1
josefson@padawan ~$ mkvirtualenv mlg [8:36:31]
New python executable in mlg/bin/python2.7
Also creating executable in mlg/bin/python
Installing setuptools, pip, wheel...done.
virtualenvwrapper.user_scripts creating /Users/josefson/virtualenvs/mlg/bin/predeactivate
virtualenvwrapper.user_scripts creating /Users/josefson/virtualenvs/mlg/bin/postdeactivate
virtualenvwrapper.user_scripts creating /Users/josefson/virtualenvs/mlg/bin/preactivate
virtualenvwrapper.user_scripts creating /Users/josefson/virtualenvs/mlg/bin/postactivate
virtualenvwrapper.user_scripts creating /Users/josefson/virtualenvs/mlg/bin/get_env_details
(mlg) josefson@padawan ~$ workon mlg [8:36:49]
(mlg) josefson@padawan ~$ which python [8:37:03]
/Users/josefson/virtualenvs/mlg/bin/python
```

Figura 2: Virtualenv – Instalação e criação.

Uma vez com um ambiente virtual criado, instalamos as bibliotecas necessárias para o desenvolvimento do projeto. Tal procedimento pode ser visto abaixo na Figura 3, onde mostra a instalação destas bibliotecas e de suas dependências.

```
josefson@padawan ~$ workon mlg
(mlg) josefson@padawan ~$ pip install --quiet ipython selenium configobj pytest pytest-cov pylint
(mlg) josefson@padawan ~$ pip freeze
appnope==0.1.0
astroid==1.3.8
configobj==5.0.6
coverage==4.0.1
decorator==4.0.4
gnureadline==6.3.3
ipython==4.0.0
ipython-genutils==0.1.0
logilab-common==1.1.0
path.py==8.1.2
pexpect==4.0.1
pickleshare==0.5
ptyprocess==0.5
py==1.4.30
pylint==1.4.4
pytest==2.8.2
pytest-cov==2.2.0
selenium==2.48.0
simplegeneric==0.8.1
six==1.10.0
traitlets==4.0.0
wheel==0.24.0
(mlg) josefson@padawan ~$
```

Figura 3: Virtualenv – Dependências.

Instalamos então o pacote `ipython` que é um shell python interativo, o `selenium` que traz a binds do Webdriver para a automação de browsers, o `configobj` para gerenciar um arquivo de configuração, o `pytest` que é um *test runner* e o `pylint` que é um linter python. Este último será utilizados para aferir a qualidade do código criado, visto que isto é essencial para a manutenção do mesmo.

O framework de testes automatizado deve ser tratado de mesma maneira que um software como visto na fundamentação, desta forma criou-se este framework dentro do repositório da própria aplicação web de forma que ambos estejam sobre o mesmo controle de versão.

As Figuras 4 e 5 nos mostra a criação do esqueleto de pastas inicial do framework e o resultado ao fim do período do estágio. O pensamento inicial é separarmos o código de testes do código da aplicação, o qual pode ser também segmentado. Em python, cada pasta é um pacote e cada arquivo um novo módulo. Desta forma, já estaremos estruturando nossos pacotes.

- Framework
  - mlg(aplicação)
    - pages
    - util
  - tests

```
josefson@padawan ~$ cd ~/Sites/mleducacao
josefson@padawan ~/Sites/mleducacao$ mkdir tests
josefson@padawan ~/Sites/mleducacao$ mkdir -p tests/framework/mlg
josefson@padawan ~/Sites/mleducacao$ mkdir -p tests/framework/tests
josefson@padawan ~/Sites/mleducacao$ mkdir -p tests/framework/mlg/pages
josefson@padawan ~/Sites/mleducacao$ mkdir -p tests/framework/mlg/util
josefson@padawan ~/Sites/mleducacao$ cd tests
josefson@padawan ~/Sites/mleducacao/tests$ tree -d
.
├── framework
│   ├── mlg
│   │   ├── pages
│   │   └── util
│   └── tests
└── tests
```

5 directories

Figura 4: Esqueleto framework.

```

└─ framework
   ├── __init__.py
   ├── config.properties
   ├── mlg
   │   ├── __init__.py
   │   ├── pages
   │   │   ├── __init__.py
   │   │   ├── base.py
   │   │   ├── billing_address.py
   │   │   ├── checkout.py
   │   │   ├── client_dashboard.py
   │   │   ├── edit_profile.py
   │   │   ├── home.py
   │   │   ├── login_register.py
   │   │   ├── product.py
   │   │   ├── recover_password.py
   │   │   ├── search.py
   │   │   ├── shipping_address.py
   │   │   └── shopping_cart.py
   │   └── util
   │       ├── __init__.py
   │       ├── config.py
   │       └── functions.py
   └── tests
       ├── __init__.py
       ├── base_testcase.py
       ├── base_testcase_login.py
       ├── checkout_test.py
       ├── edit_address_test.py
       ├── edit_profile_test.py
       ├── login_test.py
       ├── product_test.py
       ├── recover_password_test.py
       ├── search_test.py
       └── shoppingcart_test.py
5 directories, 30 files

```

Figura 5: Framework – Páginas e *Test Suites*

O importante na criação do framework é a separação entre as abstrações de páginas e os testes seguindo as orientações do *page object design*.

### 3.2 CRIAÇÃO DAS CLASSES BASE

Neste momento estamos pronto para começarmos a escrever o framework. O qual será composto de classes de páginas, as quais representam uma abstração de uma verdadeira página da aplicação e classes de *test cases*, os quais representam asserções de requisitos funcionais da aplicação. Foram então identificados 2 tipos de classes que serão frequentemente utilizadas no decorrer deste projeto, sendo assim é essencial o uso de herança em ambos os casos.

Primeiramente foi criada uma classe base para as páginas, a qual sofreu algumas incrementações ao longo do desenvolvimento e resultou no que mostra a Figura 20. Note que inicializamos um objeto passado através do construtor chamado de driver, este objeto será passado através dos *test cases*, quando nos testes estivermos chamando uma página. A variável

driver neste caso, é a entidade responsável pela navegação, seleção de elementos e interação com a página através do browser. A classe base possui 7 atributos os quais podem ser melhor visualizados abaixo.

### Class PageBase:

|                            |  |
|----------------------------|--|
| <code>_url_modifier</code> | Parte dinâmica da url para alcançar a página em específico.  |
| <code>_validate</code>     | Elemento único identificador a ser utilizado para aferir se o driver está na página correta pelo método <code>validate_page</code> . |
| <code>__init__</code>      | Método construtor, onde calcula-se a url completa da página alocando-a na variável <code>_url</code> .                               |
| <code>validate_page</code> | Método responsável por validar se o driver está de fato na página correta.   |
| <code>navigate</code>      | Método responsável por levar o driver até a página.  |
| <code>_get_base_url</code> | Método privado responsável por obter a url base da aplicação.  |
| <code>_set_url</code>      | Método privado responsável por calcular a url completa para a página em questão.   |

```

1  """
2  This module represents a base webpage which all others pages will inherit from
3  """
4  from abc import abstractmethod
5  from framework.mlg.util.config import Config
6
7
8  class BasePage(object):
9
10     """All pages inherit from this."""
11
12     _url_modifier = '' # Relative path of the page.
13     _validate = '' # Locator used to validate the page.
14
15     def __init__(self, driver):
16         self.driver = driver
17         self.config = Config()
18         self._url = self._set_url()
19
20     @abstractmethod
21     def validate_page(self):
22         """Represents an abstract method to be implemented in page classes."""
23         return
24
25     def navigate(self):
26         """Represents the hability of loading the page into a browser."""
27         self.driver.get(self._url)
28         self.validate_page()
29
30     def _get_base_url(self):
31         """Internal function to get the base url from a config file."""
32         return self.config.get('base_url')
33
34     def _set_url(self):
35         """Internal method to set the entire url based on the base url."""
36         base = self._get_base_url()
37         url = base + self._url_modifier
38         return url

```

FFigura 6: Página base.



O próximo passo é escrevermos a classe base de *test cases*, a qual é bem simples e dividida em 2 métodos. O método *setUp*, responsável pela pré-configuração dos testes e o método *tearDown*, o qual se encarregará de funções como liberar recursos que pararam de ser utilizados. Nossa classe base de testes será estendida da classe `unittest.TestCase` a qual pertence ao framework de testes da família *xUnit* e se encontra por padrão na *standard library*.

No método *setUp* introduziu-se uma forma de passar parâmetros via variáveis de ambiente, a fim de passarmos o browser e a plataforma como parâmetros. Dessa forma, podemos escolher sobre que combinação de browser e sistema operacional queremos testar nossos testes. Estes argumentos serão configurados posteriormente nas tarefas do Jenkins.

O método *setUp* é chamado antes de cada *test case*, iniciando o browser com as configurações necessárias. Os *test cases* então utilizam o *driver* para controlar páginas e realizar asserções sobre elas.

Já o método *tearDown*, é responsável pela liberação de recursos finalizando o browser após cada *test case*. A classe base de testes pode ser então vista na Figura 7.

```

1 """
2 This module server as a superclass for all the testcases we will write.
3 Here we will tweak the setUp and tearDown methods to extend their behaviour
4 through all our tests.
5 """
6 import os
7 import unittest
8 from selenium import webdriver
9 from framework.mlg.util.config import Config
10
11
12 class BaseTestCase(unittest.TestCase):
13
14     """Test Case where all tests will inherit from"""
15
16     def setUp(self):
17         """Method called immediately before every testcase in a testcase class.
18         We will use this method to setUp pre requisites for our test_cases."""
19         hub = Config.selenium_hub()
20         capabilities = {'browserName': os.environ['browser'],
21                        'platform': os.environ['platform']}
22         self.driver = webdriver.Remote(command_executor=hub,
23                                       desired_capabilities=capabilities)
24         self.driver.implicitly_wait(30)
25         self.driver.maximize_window()
26
27     def tearDown(self):
28         """Method called immediately after every testcase results have been
29         recorded. This methos is called even if the testcase raises an error"""
30         self.driver.quit()

```

Figura 7: Classe base de *test cases*.



Uma vez que foram criadas as classes base de nosso framework, as quais serão herdadas por nossas classes de páginas e nossas classes de testes, podemos então começar a escrita destas classes.

### 3.3 ENTENDENDO O PAGE OBJECT PATTERN

O menor e mais conciso exemplo foi a criação da página de login e registro da aplicação. Aqui mostraremos alguns conceitos básicos que foi utilizado no decorrer da criação de todas as páginas. Primeiramente é preciso entender o conceito de *página* abordado pelo page object design pattern. Na Figura 8 é mostrada a página de login e registro, a qual possui muitos elementos html. Possuí um *header* com elementos de navegação, logo abaixo possui também um campo de busca paralelo ao carrinho de compras, posteriormente uma outra seção de navegação para seções de produtos e só abaixo temos a parte de login e registro, temos ainda abaixo e não mostrado na imagem, o *footer* da página.

Poderíamos criar uma página com métodos de interação com todos os elementos mostrados aqui, testando todos em questão. Entretanto, isso seria desaconselhável por algumas razões.

- Em primeiro lugar estamos escrevendo testes para validações de requisitos funcionais. Desta forma focaremos nossos esforços em primeiramente seguir o *money-path* da aplicação. Quais são os requisitos necessários para a aplicação continuar gerando renda para nosso cliente?
- Em segundo lugar, de acordo com o *object design pattern* e o *DRY*, evitamos o máximo a repetição de código. Desta forma, o correto foi separarmos os elementos em seções os quais caracterizam por si só uma “página objeto”.

Assim, cada seção agrupada pela cor vermelha na Figura 8 caracteriza uma página e caso necessário serão criadas páginas para elas. Desta forma, evitamos a duplicidade de código maximizando a manutenção.

Caso queiramos em algum teste acessar o *header*, basta importarmos a página *header* para o teste e passarmos o driver usado. Uma vez que *header* está desvinculada a qualquer página e ainda assim presente em todas, tomamos ele como um módulo que compõe uma página real. Ou seja, sempre que identificarmos um grupo de elementos único que se repetem

em várias páginas da aplicação eles provavelmente enfrentam o *page object design pattern*.

Sendo assim, tomaremos apenas a terceira seção em vermelho como sendo nossa página de login e registro. Desta forma, iniciaremos a criação de nossa página de testes com esta decisão em mente.

CANAIS DE ATENDIMENTO | POR QUE COMPRAR NA MLG? | AULA GRÁTIS | BLOG

ÁREA DO ALUNO

MLG | educação

O que você quer estudar?

Pesquisa...

LOGIN / REGISTRO

CARRINHO / R\$ 0,00

Todos os Cursos | Recursos Humanos | Manufatura & Sistemas Lean | PNL Neurolinguística

MAIS CURSOS

### MINHA CONTA

#### ENTRAR

Nome de usuário ou e-mail \*

Senha \*

ENTRAR

☐ Lembre-me

[Perdeu sua senha?](#)

#### REGISTRAR

Endereço de e-mail \*

Senha \*

REGISTRAR

Figura 8: Página de login.

### 3.4 ESCRIVENDO UMA PÁGINA

Pensando sobre a perspectiva de um usuário, iremos decompor as ações necessárias para a realização de um login. Estando com o browser aberto, necessitamos navegar até a página em questão, depois digitarmos nossas credenciais e clicamos em entrar ou apertamos enter. Utilizaremos estas ações como métodos da página a serem utilizados como interfaces em nossos testes utilizando uma linguagem o mais próxima possível do negócio.

Além disso, necessitamos dos identificadores de elementos html que estaremos utilizando nesta página.

- Campos de e-mail.
- Campos de senha.
- Botões de entrar e registrar.
- Checkbox de Lembrar.

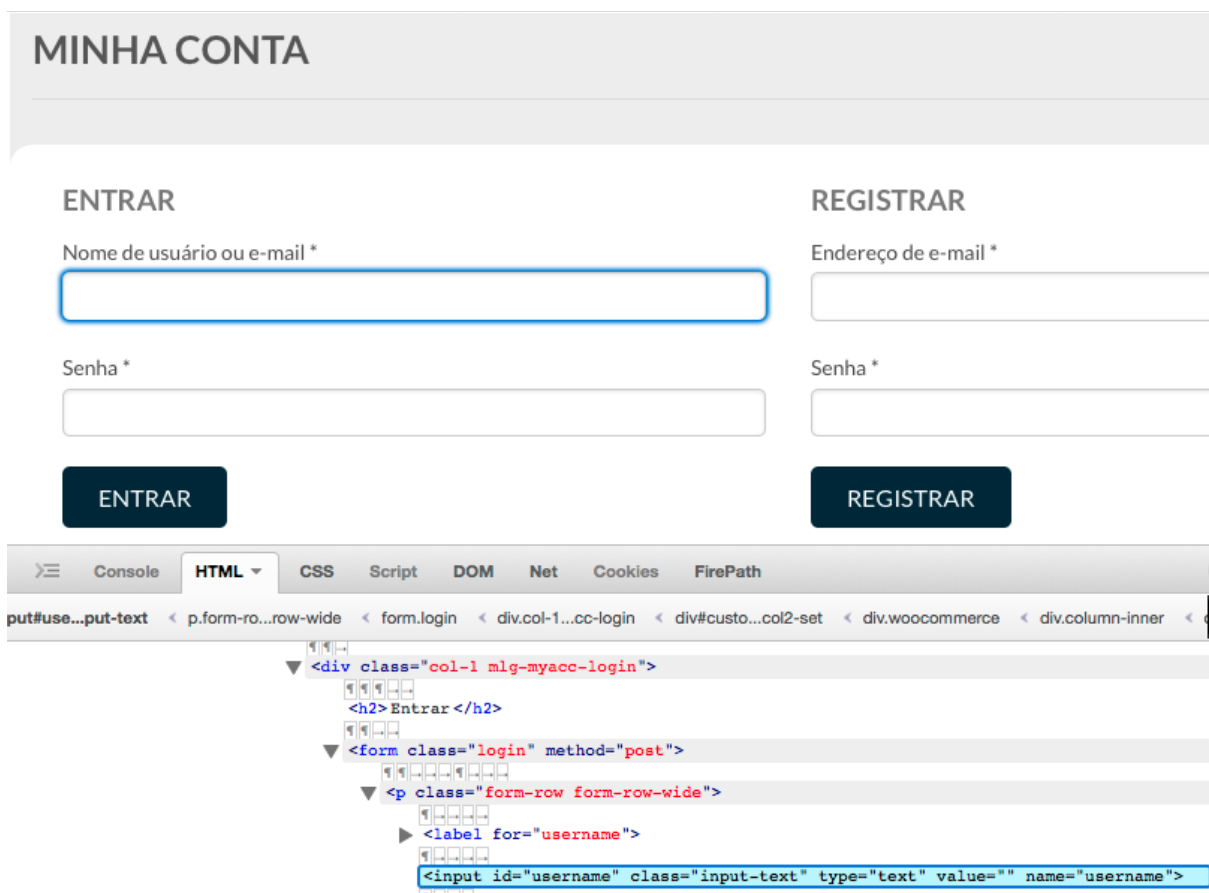


Figura 9: Localizando um Identificador.

Tais identificadores estão no código html da página e os buscaremos utilizando o browser Mozilla Firefox com dois plugins adicionais: o Firebug e o Firepath. A Figura 9 mostra a inspeção de um destes elementos através da extensão Firebug. Neste momento é importante frisar que a ferramenta selenium consegue identificar elementos através de alguns atributos, sendo eles:

- ID
- CLASS\_NAME
- CSS\_SELECTOR
- XPTAH
- NAME
- TAG\_NAME
- LINK\_TEXT
- PARTIAL\_LINK\_TEXT

Isto é importante pois devemos priorizar os seletores escolhidos de forma a deixarmos nosso framework o mais resiliente possível a eventuais mudanças na aplicação. Então sempre que pudemos utilizamos os seletores ID, CLASS\_NAME e NAME, casos os elementos que precisamos selecionar não possuam tais identificadores ou caso, tais identificadores estejam sendo compartilhados com outros elementos, utilizamos outros identificadores.

Em nossa página de login, a maioria dos elementos possuem identificadores únicos como o mostrado na Figura 9 , dessa forma anotamos todos os identificadores a serem

utilizados em nossa página. Agora possuímos as três peças fundamentais na criação de uma página: as ações que um usuário realizará sobre a página, e os identificadores necessários para o selenium identificar e interagir com o elemento. Uma vez que tenhamos essas peças podemos então iniciar a construção de nossa página, como mostrado nas Figuras 10 e 11.

Adicionaremos os identificadores dos elementos como atributos estáticos da página, visto que os mesmos não mudam em tempo de execução. Depois temos nosso construtor, o qual chama o construtor da superclasse. Em seguida temos uma implementação do método abstrato de validação de página de nossa superclasse, o qual será invocado nos testes antes de interagirmos com elementos da página. Logo depois, temos os principais métodos da página, o login e o register os quais são responsáveis por realizar as interações com a página para desempenhar respectivamente as ações de login e o registro na aplicação.

```

1  # -*- coding: utf-8 -*-
2  """This module is responsible for the abstraction of the login and register
3  functionality of the application."""
4  from framework.mlg.pages.base import BasePage
5  from framework.mlg.pages.base import InvalidPageException
6  from framework.mlg.pages.client_dashboard import ClientDashboard
7  from framework.mlg.pages.recover_password import RecoverPassword
8  from selenium.webdriver.common.by import By
9
10
11  class LoginRegister(BasePage):
12
13      """This class represents the page related to login and registration."""
14
15      _url_modifier = '/minha-conta'
16      _title = 'Minha conta | MLG Educação'
17      _log_name_loc = 'username'
18      _log_passwd_loc = 'password'
19      _log_remember = 'rememberme'
20      _log_forgot = 'Perdeu sua senha?' # LINK_TEXT
21      _reg_name_loc = 'reg_email'
22      _reg_passwd_loc = 'reg_password'
23      _error_msg_loc = '.alert-box.alert.animated.fadeIn'\
24                      '.woocommerce-error' # CSS
25
26      def __init__(self, driver):
27          super(LoginRegister, self).__init__(driver)
28
29      def validate_page(self):
30          """Checks for element and its text in order to validate the page."""
31          element = self.driver.find_element(By.CSS_SELECTOR, '#content>h1')
32          if element.text.encode('utf-8') != 'MINHA CONTA':
33              raise InvalidPageException("Loading a non expected page")
34          else:
35              return True
36
37      def login(self, user, passwd):
38          """This method will input the user and passwd credentials into the form
39          and click the login button if it succeeds it will return the
40          ClientDashboard page."""
41          self.type_username_login(user)
42          self.type_password_login(passwd)
43          locator = self._log_passwd_loc
44          element = self.driver.find_element(By.ID, locator)
45          element.send_keys('\n')
46          return ClientDashboard(self.driver)

```

Figura 10: Página LoginRegister 1.

```

48     def register(self, user, passwd):
49         """This method will input the user and passwd credentials into the form
50         and click the register button, if it succeeds it will return the
51         ClientDashboard page."""
52         self.type_username_reg(user)
53         self.type_password_reg(passwd)
54         locator = self._reg_passwd_loc
55         element = self.driver.find_element(By.ID, locator)
56         element.send_keys('\n')
57         return ClientDashboard(self.driver)
58
59     def _type(self, locator, text):
60         """Internal method for typing a text given an ID locator."""
61         element = self.driver.find_element(By.ID, locator)
62         element.clear()
63         element.send_keys(text)
64
65     def type_username_login(self, user):
66         """Type the username at the login section."""
67         self._type(self._log_name_loc, user)
68
69     def type_password_login(self, passwd):
70         """Type the password at the login section."""
71         self._type(self._log_passwd_loc, passwd)
72
73     def type_username_reg(self, user):
74         """Type the username at the registration section."""
75         self._type(self._reg_name_loc, user)
76
77     def type_password_reg(self, passwd):
78         """Type the password at the registration section."""
79         self._type(self._reg_passwd_loc, passwd)
80
81     def check_rememberme(self):
82         """Check the 'remember me' function at the login section."""
83         locator = self._log_remember
84         element = self.driver.find_element(By.ID, locator)
85         element.click()
86
87     def recover_password(self):
88         """This method clicks the recover function and returns its page."""
89         locator = self._log_forgot
90         element = self.driver.find_element(By.LINK_TEXT, locator)
91         element.click()
92         return RecoverPassword(self.driver)
93
94     def error_message(self):
95         """This method returns an error element to test assertions."""
96         locator = self._error_msg_loc
97         element = self.driver.find_element(By.CSS_SELECTOR, locator)
98         return element

```

Figura 11: Página LoginRegister 2.

De forma análoga, foram criadas 12 abstrações de páginas: base.py, home.py, login\_register.py, client\_dashboard.py, edit\_profile.py, recover\_password.py, search.py, shipping\_address.py, billing\_address.py, product.py, shopping\_cart.py e checkout.py.

### 3.5 ESCRIVENDO UM TEST CASE

Escrevemos então uma classe `TestCase` para cada grupo de testes referente àquela página, no caso da página de login queremos testar a resposta a um login válido, a resposta a um login inválido e o registro de um novo cliente.

Herdando a classe `BaseTestCase`, temos por herança os métodos `setUp` e `tearDown`, sendo o `setUp` sobrescrito e invocando o da superclasse de forma a complementar o código que seria repetido nos três *test cases* em questão. *Test cases* estes que são os métodos iniciados com a palavra *test*, requisito necessário para sua própria identificação quando, visto que uma classe pode ter métodos auxiliares que não são *test cases*.

Neste trabalho foram desenvolvidos 8 *test suites* e 2 classes bases das quais eles poder herdar: `base_testcase.py`, `base_testcase_login.py`, `login_test.py`, `recover_password_test.py`, `edit_profile_test.py`, `edit_address_test.py`, `search_test.py`, `product_test.py`, `shoppingcart_test.py`, `checkout_test.py`.

A Figura 12 mostra o exemplo do *test suite* `login_test.py`, o qual contém 3 *test cases*. Ao todo, distribuído nos 8 *test suites*, foram criados 22 *test cases*:

|  |                                       |
|--|---------------------------------------|
| <code>test_invalid_login</code>            | <code>test_checkout_navigation</code> |
| <code>test_valid_login</code>              | <code>test_login_checkout</code>      |
| <code>test_registration</code>             | <code>test_add_qty</code>             |
| <code>test_recover_from_login_page</code>  | <code>test_rem_qty</code>             |
| <code>test_recover_password</code>         | <code>test_free_class</code>          |
| <code>test_invalid_billing_address</code>  | <code>test_buy</code>                 |
| <code>test_invalid_shipping_address</code> | <code>test_change_qty_product</code>  |
| <code>test_valid_billing_address</code>    | <code>test_increment_product</code>   |
| <code>test_valid_shipping_address</code>   | <code>test_decrement_product</code>   |
| <code>test_edit_profile_valid</code>       | <code>test_remove_product</code>      |
| <code>test_search_product</code>           | <code>test_show_product</code>        |



```

1 """
2 This module is responsible for testing the login and registration
3 functionality.
4 """
5 import unittest
6 from framework.mlg.util.config import Config
7 from framework.mlg.util.functions import new_email
8 from framework.tests.base_testcase import BaseTestCase
9 from framework.mlg.pages.login_register import LoginRegister
10
11
12 class LoginTest(BaseTestCase):
13     """This class tries to agregate testcases related to the login page."""
14
15     def setUp(self):
16         """Fixture of our login testcases. Extending our super SetUp
17         in order to provide less duplicated code across testcases."""
18         super(LoginTest, self).setUp()
19         self.page = LoginRegister(self.driver)
20         self.page.navigate()
21
22     def test_invalid_login(self):
23         """Testcase for invalid login. Here we check for nice error message."""
24         self.page.login('user', 'passwd')
25         error = self.page.error_message()
26         self.assertIsNotNone(error)
27         self.assertTrue(error.is_displayed())
28
29     def test_valid_login(self):
30         """Testcase for valid login. Here we check for the dashboard page
31         as a result of logging in with a valid login."""
32         self.page.login('user', 'passwd')
33         email = Config.get('client_email')
34         passwd = Config.get('client_passwd')
35         dashboard = self.page.login(email, passwd)
36         self.assertTrue(dashboard.validate_page())
37
38     def test_registration(self):
39         """Testcase for registration. Here we try to register a new user
40         and check for the dashboard resulting page."""
41         email = new_email()
42         passwd = email
43         dashboard = self.page.register(email, passwd)
44         self.assertTrue(dashboard.validate_page())

```

Figura 12: LoginTest – Test Cases.



### 3.6 SELENIUM GRID

Selenium provê o Selenium Standalone Server, um executável java que pode funcionar em 2 modos:

1. Standalone Server, no qual ele cria um webserver responsável por escutar uma porta esperando por requisições.
2. Selenium Grid, no qual pode-se iniciar o executável passando sua *role*, a qual se divide entre *hub* e *node*.

Através da segunda opção pudemos configurar uma rede de recursos sobre os quais queremos que nossos testes sejam testados. Basicamente estes recursos são configurados por um arquivo de configuração lido na inicialização do Selenium Standalone Server, o qual foi configurado como um serviço.

A idéia aqui é possuímos um *hub*, o qual se mantém atualizado sobre quais recursos existem e estão disponíveis. Nossos testes farão requisições de recursos a este *hub*, perguntando se uma combinação de sistema operacional e browser existe e está disponível.

Uma vez que tal requisição for atendida nossos testes passam a se comunicar diretamente com o nó através da camada de rede, enviando comandos que são interpretados pelo Selenium em execução naquele nó de forma que o browser e os testes ocorrem naquele nó e por fim ele retorna os resultados dos testes. Tal esquema pode ser melhor visualizado na Figura 13.

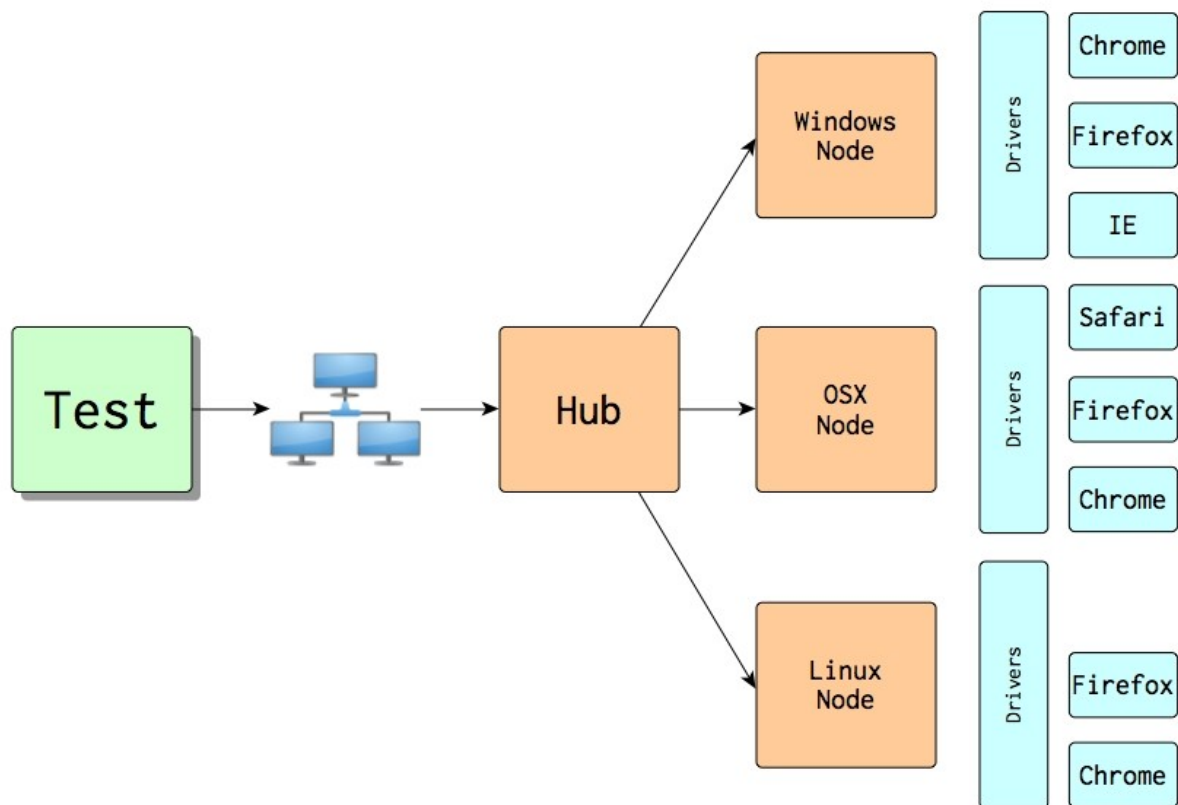


Figura 13: Selenium grid – Schema.

Uma vez tendo baixado o selenium-server.jar, este foi executado de duas formas diferentes.

- 1) `java -jar selenium-server.jar -role hub`
- 2) `java -jar selenium-server.jar -role node -configNode node.json`

Ambas foram configuradas como serviço, o qual é iniciado na inicialização do sistema operacional. A única diferença é que utilizamos a primeira forma de inicialização na mesma máquina onde terá o Jenkins instalado. Desta forma a mesma máquina tanto será responsável pelo gerenciamento das tarefas do Jenkins, como também pela inicialização do *selenium hub*, o qual uma vez iniciado fica a espera de registros de novos nós.

A Figura 14 nos mostra o processo de inicialização de um *hub* o qual está a espera de registro de novos nós para compor o grid de recursos.

```

(mlg) josefson@padawan ~ java -jar selenium-server.jar -role hub [14:21:19]
14:23:21.413 INFO - Launching Selenium Grid hub
2015-10-21 14:23:24.294:INFO:main: Logging initialized @6456ms
14:23:24.349 INFO - Will listen on 4444
14:23:24.414 INFO - Will listen on 4444
2015-10-21 14:23:24.420:INFO:osjs.Server:main: jetty-9.2.z-SNAPSHOT
2015-10-21 14:23:24.542:INFO:osjs.ContextHandler:main: Started o.s.j.s.ServletContextHandler@612fc6eb{/,null,AVAILABLE}
2015-10-21 14:23:24.727:INFO:osjs.ServerConnector:main: Started ServerConnector@7a30die6{HTTP/1.1}{0.0.0.0:4444}
2015-10-21 14:23:24.728:INFO:osjs.Server:main: Started @6890ms
14:23:24.739 INFO - Nodes should register to http://192.168.1.2:4444/grid/register/
14:23:24.740 INFO - Selenium Grid hub is up and running

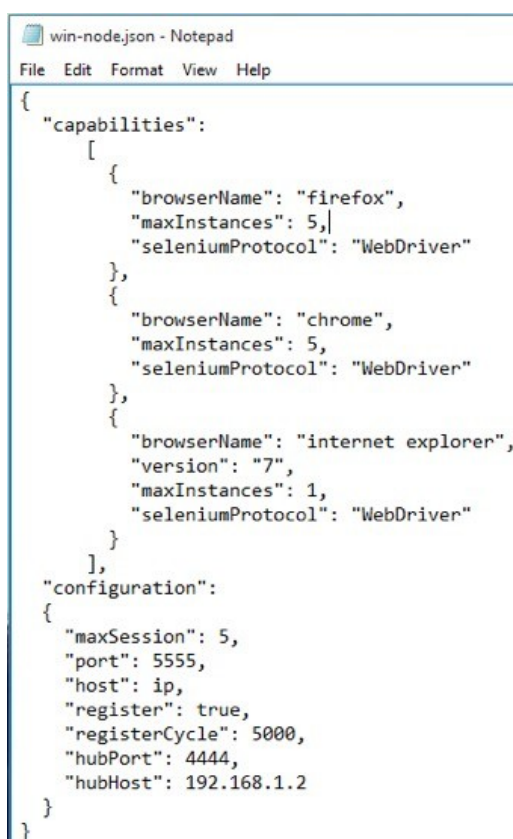
```

Figura 14: Selenium grid – Hub.

De forma análoga, utilizamos o mesmo conceito para a inicialização de nossos nós de recursos, a única diferença é que estes se iniciam junto a um arquivo de configuração. O qual é responsável por:

- Listar os browsers disponíveis na máquina.
- Listar as versões de cada browser. Caso houver mais de uma versão do mesmo. browser é necessário o caminho para cada executável.
- Listar quantas instâncias do browser podem ser executadas ao mesmo tempo.
- Listar quantas sessões podem estar ativas ao mesmo tempo.

A Figura 15 nos mostra o exemplo de um dos arquivos de configuração utilizados em uma das máquinas.



```

win-node.json - Notepad
File Edit Format View Help
{
  "capabilities":
  [
    {
      "browserName": "firefox",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver"
    },
    {
      "browserName": "chrome",
      "maxInstances": 5,
      "seleniumProtocol": "WebDriver"
    },
    {
      "browserName": "internet explorer",
      "version": "7",
      "maxInstances": 1,
      "seleniumProtocol": "WebDriver"
    }
  ],
  "configuration":
  {
    "maxSession": 5,
    "port": 5555,
    "host": "ip",
    "register": true,
    "registerCycle": 5000,
    "hubPort": 4444,
    "hubHost": "192.168.1.2"
  }
}

```

Figura 15: Selenium grid – Configuração de nó.

A Figura 16 nos mostra a inicialização de um nó a qual se dá pela execução do seguinte comando, o qual fora configurado para ser executado na inicialização do sistema operacional.

*java -jar selenium-server.jar -role node -nodeConfig /path/to/config.json*

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\josefson>java -jar selenium-server-standalone-2.48.2.jar -role node -nodeConfig win-node.json
16:28:27.501 INFO - Launching a Selenium Grid node
16:28:28.392 INFO - Java: Oracle Corporation 25.65-b01
16:28:28.392 INFO - OS: Windows 10 10.0 x86
16:28:28.392 INFO - v2.48.0, with Core v2.48.0. Built from revision 41bccdd
16:28:28.454 INFO - Driver class not found: com.opera.core.systems.OperaDriver
16:28:28.454 INFO - Driver provider com.opera.core.systems.OperaDriver is not registered
16:28:28.517 INFO - Selenium Grid node is up and ready to register to the hub
16:28:28.564 INFO - Starting auto registration thread. Will try to register every 5000 ms.
16:28:28.564 INFO - Registering the node to the hub: http://192.168.1.2:4444/grid/register
16:28:28.782 INFO - The node is registered to the hub and ready to use
```

Figura 16: Selenium grid – Nó.

Verifica-se então a inicialização do ó, o qual diz-se já ter se registrado com o hub, o que pode ser comprovado na visualização od console ou log do hub, como mostrado na última linha da Figura 17.

```
(mlg) josefson@padawan ~ java -jar selenium-server.jar -role hub [14:21:19]
14:23:21.413 INFO - Launching Selenium Grid hub
2015-10-21 14:23:24.294:INFO:main: Logging initialized @6456ms
14:23:24.349 INFO - Will listen on 4444
14:23:24.414 INFO - Will listen on 4444
2015-10-21 14:23:24.420:INFO:osjs.Server:main: jetty-9.2.z-SNAPSHOT
2015-10-21 14:23:24.542:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextHandler@612fc6eb(/,null,AVAILABLE)
2015-10-21 14:23:24.727:INFO:osjs.ServerConnector:main: Started ServerConnector@7a30d1e6(HTTP/1.1){0.0.0.0:4444}
2015-10-21 14:23:24.728:INFO:osjs.Server:main: Started @6890ms
14:23:24.739 INFO - Nodes should register to http://192.168.1.2:4444/grid/register/
14:23:24.740 INFO - Selenium Grid hub is up and running
15:28:33.561 INFO - Registered a node http://192.168.1.12:5555
```

Figura 17: Selenium grid – Registro de nó.

Pode-se também verificar o estado do grid pelo endereço <http://hub-ip/grid/console>, no nosso caso, o resultado com os vários nós rodando é o apresentado na Figura 18.

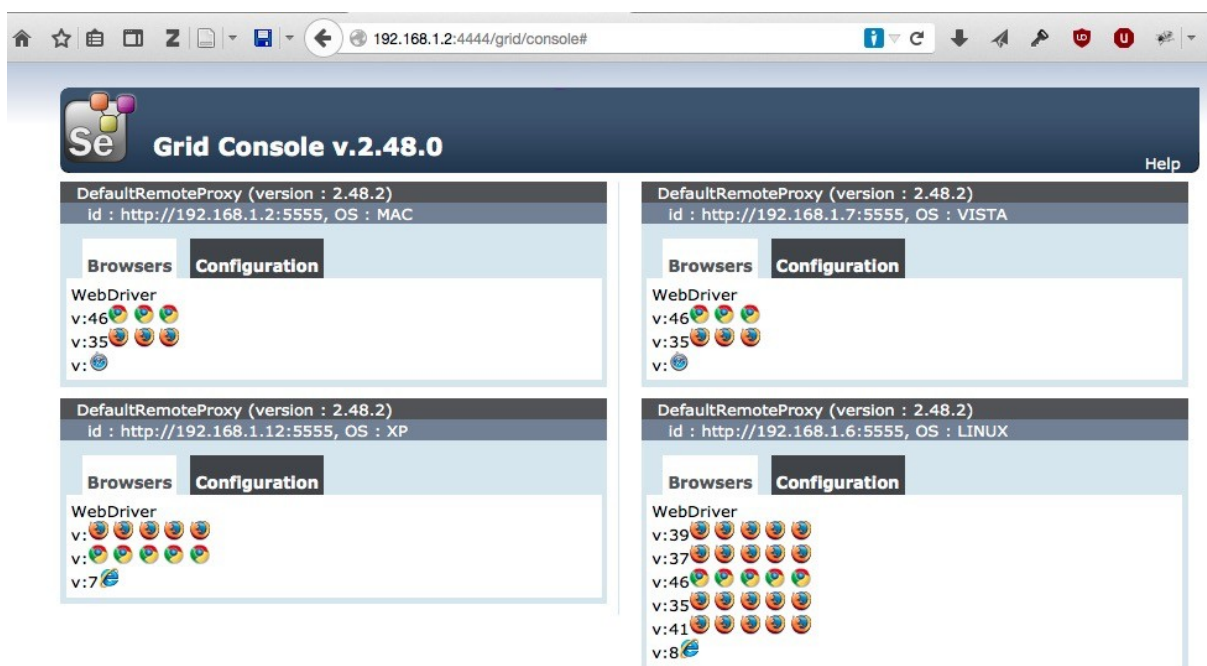


Figura 18: Selenium grid – Recursos

### 3.7 JENKINS

O Jenkins foi utilizado em nosso trabalho basicamente como um agregador de tarefas, as quais foram configuradas para:

- Monitorar o *branch* develop dos repositórios dos desenvolvedores na rede.
- A partir de algum *commit*, disparar a execução da tarefa:
  - Réplica do código base do repositório que disparou o evento.
  - Construção do ambiente de testes.
  - Execução de testes.

O Jenkins será então o responsável pela execução automática e regressiva da suite de testes desenvolvida. Se encarregando da execução, controle de ambiente de testes e do vínculo entre um *commit* e o resultado dos testes ocasionado por ele.

Depois de baixado o `jenkins.war` do site oficial, configuramos sua execução como serviço de sistema. Depois foi necessárias a instalação de alguns plugins, os quais puderam ser instalados através da própria interface do Jenkins.

Após iniciado serviço, um webserver estará rodando na porta 7777 da máquina servidor. A partir deste endereço pode-se acessar a interface Jenkins para maiores configurações, como pode ser visto na Figura 19.

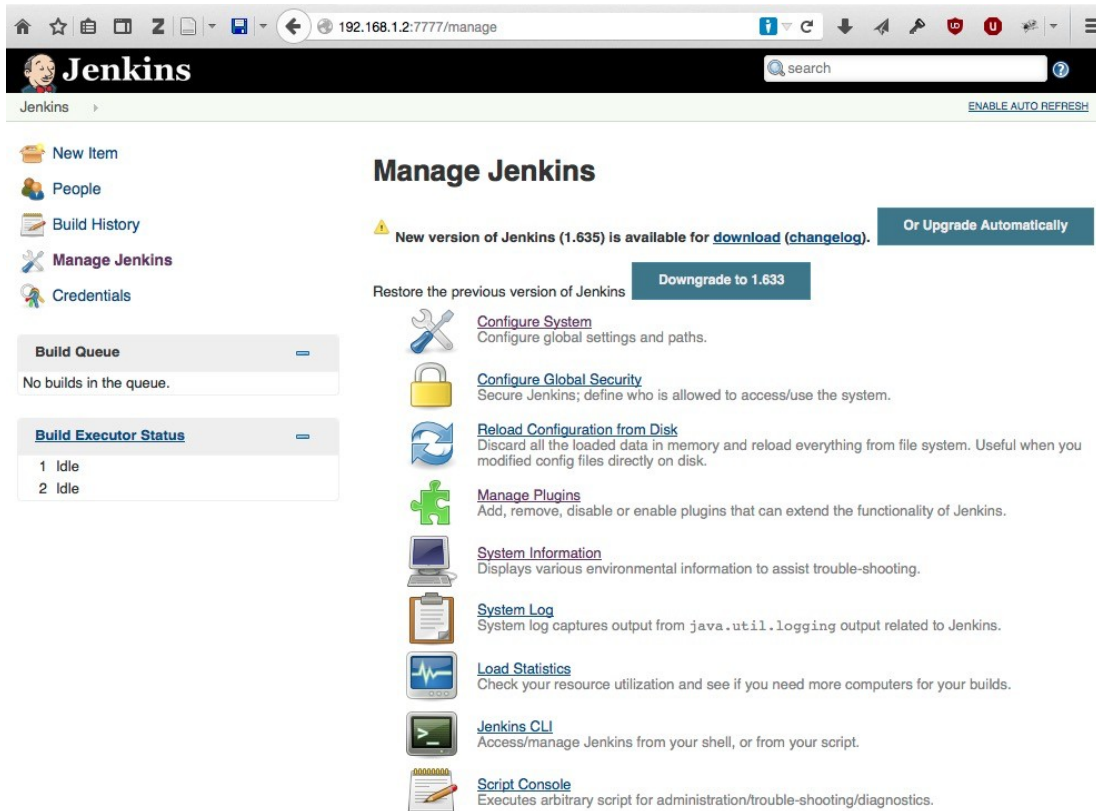


Figura 19: Jenkins – Página de configuração.

Acessando o menu *Manage Jenkins*>>*Manage Plugins*>>*Available* pudemos buscar e instalar o Git Plugin, o qual nos permite a utilização de repositórios git como um SMC.

O próximo passo foi a configuração geral do Jenkins, que é acessada pelo menu *Manage Jenkins*>>*Configure System*, onde indicamos algumas configurações principais:

- # of executors – Configura o número de workers que podem executar tarefas em paralelo.
- Path to Git executable – Configura o caminho para o git a ser utilizado.
- Shell executable – Configura o caminho do shell a ser utilizado.

Tais configurações podem ser melhor visualizada na Figura 20.

Figura 20: Jenkins – Configurações Gerais



Uma vez que tivemos nosso Jenkins configurado, pudemos de fato adicionar novos itens. Um item do Jenkins é um projeto ou uma tarefa. A Figura 21 mostra a adição de uma nova tarefa do tipo Freestyle ao nosso ambiente Jenkins.

Jenkins > All >

**New Item**

People

Build History

Manage Jenkins

Credentials

**Build Queue**

No builds in the queue.

**Build Executor Status**

1 Idle

2 Idle

Item name:

☒ **Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

☐ **Maven project**  
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

☐ **External Job**  
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

☐ **Multi-configuration project**  
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

☐ **Copy existing Item**  
Copy from:

OK

Figura 21: Jenkins – Nova tarefa

Precisamos então vincular esta tarefa ao repositório git da aplicação web que será “monitorada”. Como mostrado na Figura 22, selecionamos o SCM utilizado e indicamos o caminho do repositório, bem como os *branches* que iremos utilizar.

**Source Code Management**

☐ None

☐ CVS

☐ CVS Projectset

☒ **Git**

**Repositories**

Repository URL:

Credentials:

**Branches to build**

Branch Specifier (blank for 'any'):

**Repository browser**

**Additional Behaviours**

☐ Subversion

Figura 22: Configurando tarefa – SMC.

Depois precisamos configurar como este trabalho será executado, o que pode ser verificado na Figura 23. O ideal é que ele seja executado automaticamente dado um novo *commit* no *branch* configurado, mas infelizmente o Jenkins por padrão tenta de tempo em tempo verificar o estado do repositório, o que não é uma solução ideal. A fim de contornar isso e termos uma resposta imediata após os *commits* adicionamos o arquivo post-commit no diretório `.git/hooks/` da aplicação web com o conteúdo mostrado na Figura 24. Desta forma, sempre que houver um novo *commit*, o git notificará o Jenkins, o qual por sua vez iniciará a

execução de todas as tarefas que estejam monitorando aquele diretório. Esta mensagem de notificação pode ser verificada na Figura 25, logo após o commit.



Figura 23: Configurando tarefa – Gatilho.

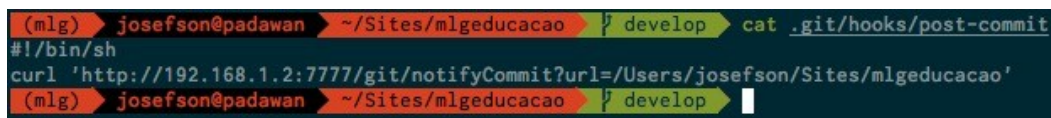


Figura 24: Configurando tarefa – Notificação após commit.

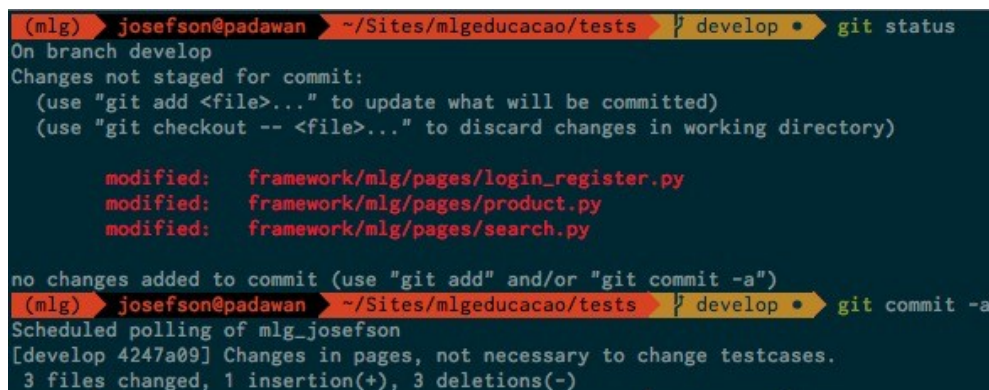


Figura 25: Notificação pós commit

Tendo configurado nosso processo de ativação de tarefa, pudemos então configurar de fato o que é a tarefa, ou seja a *build*. Adicionamos um *build step* para execução de *shell*, o qual irá executar as seguintes ações:

- Remover o ambiente python criado pela build anterior.
- Criar um novo ambiente python para esta build.
- Ativar o ambiente.
- Instalar as dependências necessárias.
- Configurar os parâmetros de teste para esta tarefa.
- Executar os testes gerando um relatório xml.

Isto foi configurado conforme mostrado na Figura 26.





Figura 26: Configurando tarefa - Build

Depois configuramos uma Ação Pós Build para anexar o relatório xml gerado à página de resultados da build provida pelo Jenkins. A Figura 27 mostra a configuração desta ação.

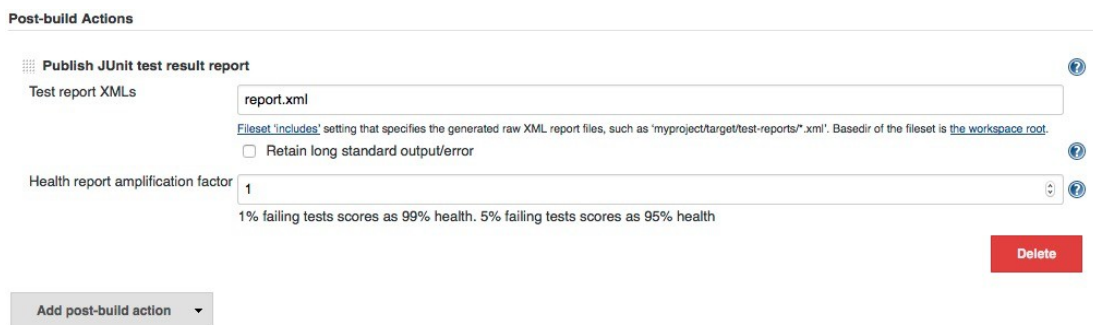
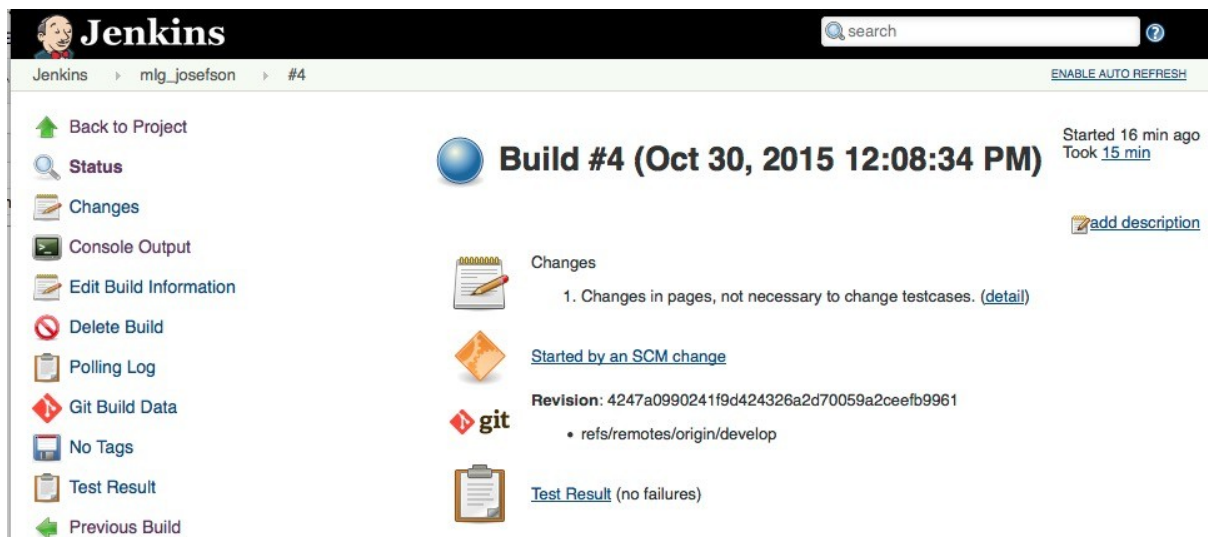


Figura 27: Configurando tarefa - Ação pós build

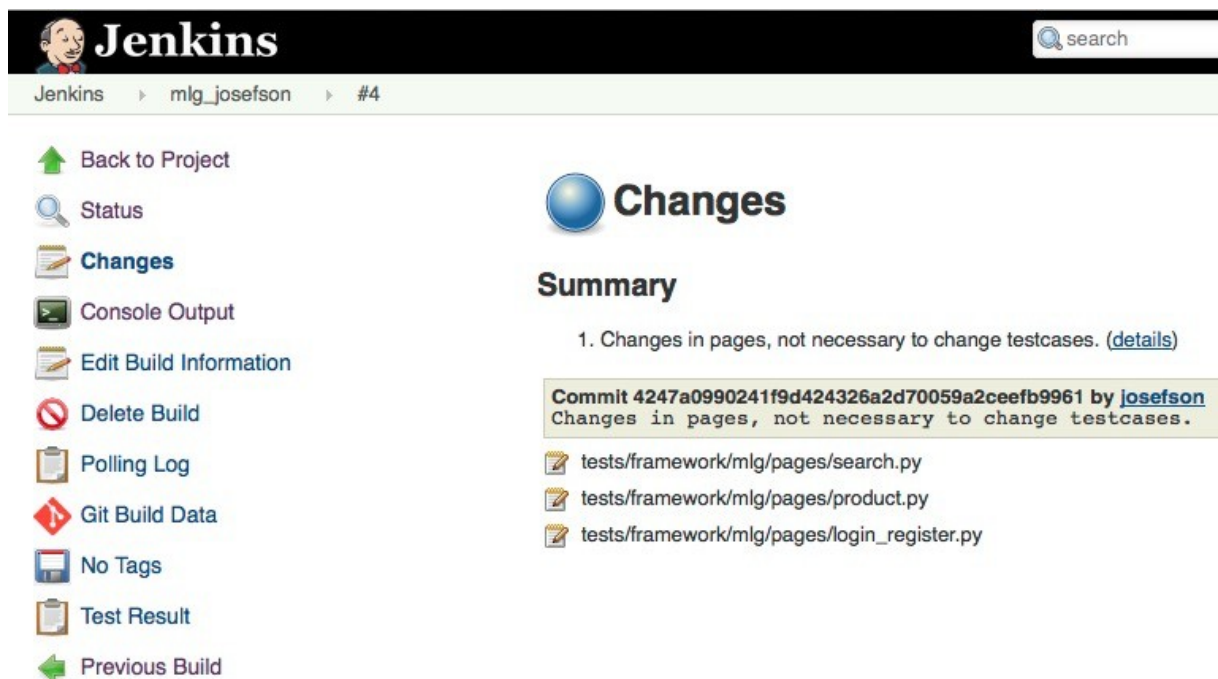
Esta configuração foi disponibilizada para todos os desenvolvedores que trabalhavam no projeto mlg. Dessa forma, sempre que havia um novo *commit* no *develop branch* em seus repositórios locais, o servidor Jenkins era notificado, o que ativava a tarefa configurada para aquele desenvolvedor. A tarefa fazia então o clone ou pull das alterações em relação ao último *commit*, depois executava os testes sobre a aplicação.

Uma vez que a execução dos testes era concluída, o relatório que fora gerado pelo *test-runner* era então consumido pelo Jenkins e vinculado àquela build. O Jenkins então agrega a mensagem de *commit* e o resultado dos testes à build, como pode ser visualizado nas Figuras 28 e 29.



The screenshot shows the Jenkins web interface for Build #4. The top navigation bar includes the Jenkins logo, a search bar, and the breadcrumb 'Jenkins > mlg\_josefson > #4'. A left sidebar contains links: 'Back to Project', 'Status', 'Changes', 'Console Output', 'Edit Build Information', 'Delete Build', 'Polling Log', 'Git Build Data', 'No Tags', 'Test Result', and 'Previous Build'. The main content area is titled 'Build #4 (Oct 30, 2015 12:08:34 PM)' and shows 'Started 16 min ago' and 'Took 15 min'. It includes a 'Changes' section with one item: '1. Changes in pages, not necessary to change testcases. (detail)'. Below this, it states 'Started by an SCM change' and 'Revision: 4247a0990241f9d424326a2d70059a2ceefb9961' with a 'git' icon and 'refs/remotes/origin/develop'. A 'Test Result' link indicates '(no failures)'. An 'add description' link is also present.

Figura 28: Build - Resumo



The screenshot shows the 'Changes' view for Build #4. The top navigation bar is identical to the previous screenshot. The left sidebar is the same. The main content area is titled 'Changes' and 'Summary'. It lists '1. Changes in pages, not necessary to change testcases. (details)'. A commit box shows 'Commit 4247a0990241f9d424326a2d70059a2ceefb9961 by josefson' with the message 'Changes in pages, not necessary to change testcases.'. Below the commit, three files are listed: 'tests/framework/mlg/pages/search.py', 'tests/framework/mlg/pages/product.py', and 'tests/framework/mlg/pages/login\_register.py'.

Figura 29: Build - Commit responsável

Podemos ainda ver os *test suites* com mais detalhes, sabendo quais foram executados e quantos *test cases* cada *test suite* contém. A figura 30 nos mostra o resumo de *test suites* executados, especificando a quantidade de *test cases* e quantos destes passaram, falharam ou foram puladas durante a execução dos testes.

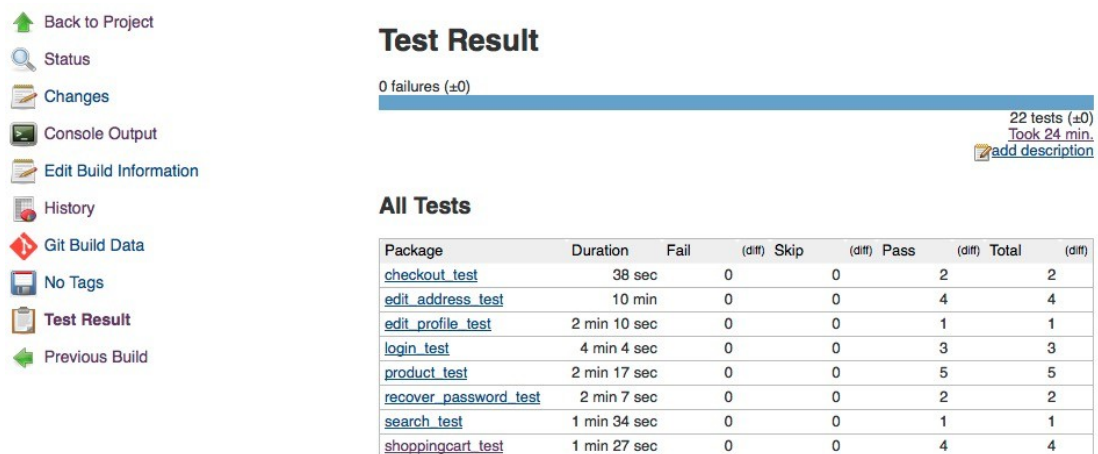


Figura 30: Build - Resultado de testes

Podemos ainda ver quais *test cases* estão contidos em cada *test suite*, o que geralmente é feito em caso de falhas a fim de averiguar o ocorrido e tomar as medidas necessárias. A Figura 31 mostra a exibição dos *test cases* contidos em em um *test suite*.

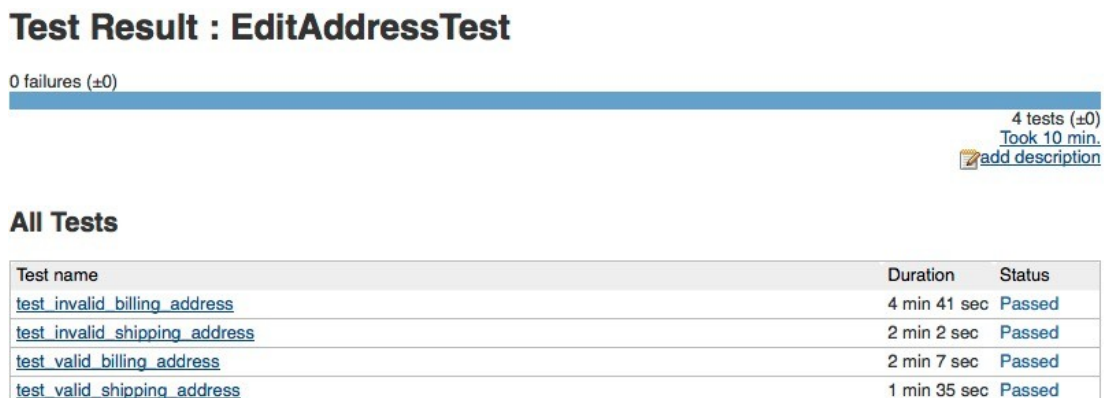


Figura 31: Build -Test Suite – Test Cases

Em casos de falhas, cada *test case* que falhou guardará neste relatório tanto as pertinentes mensagens de erro quanto o *stacktrace*. Ambos são utilizados para averiguar a causa da falha do teste. A Figura 32 mostra um exemplo de relatório de erro em um *test case*, o qual se dá origem na falha de localização de um elemento, mais especificamente o campo '*username\_*' na página de login. Ao verificarmos o *commit* relacionado a esta build pudemos ver o que ocasionou o erro e reverter o erro.

## Regression

login\_test.LoginTest.test\_invalid\_login (from pytest)

Failing for the past 1 build (Since  #6)

Took 1 min 24 sec.

 [add description](#)

### Error Message

```
NoSuchElementException: Message: Unable to locate element: {"method":"id","selector":"username_"} For
documentation on this error, please visit: http://seleniumhq.org/exceptions/no\_such\_element.html
Build info: version: '2.48.2', revision: '41bccdd', time: '2015-10-09 19:59:12' System info: host:
'josefson-pc', ip: '192.168.56.1', os.name: 'Windows 10', os.arch: 'x86', os.version: '10.0',
java.version: '1.8.0_65' Driver info: driver.version: unknown Screenshot: available via screen
Stacktrace:      at <anonymous class>.FirefoxDriver.prototype.findElementInternal (file:///C:/Users
/josefson/AppData/Local/Temp/anonymous6328684479439410119webdriver-profile/extensions
/fxdriver@googlecode.com/components/driver-component.js:10659)      at <anonymous
class>.fxdriver.Timer.prototype.setTimeout/<.notify (file:///C:/Users/josefson/AppData/Local
/Temp/anonymous6328684479439410119webdriver-profile/extensions/fxdriver@googlecode.com/components
/driver-component.js:621)
```

### Stacktrace

```
self = <framework.tests.login_test.LoginTest testMethod=test_invalid_login>

  def test_invalid_login(self):
    """Testcase for invalid login. Here we check for nice error message."""
>     self.page.login('user', 'passwd')

tests/framework/tests/login_test.py:24:
tests/framework/mlg/pages/login_register.py:42: in login
  self.type_username_login(user)
tests/framework/mlg/pages/login_register.py:68: in type_username_login
  self._type(self._log_name_loc, user)
tests/framework/mlg/pages/login_register.py:62: in _type
  element = self.driver.find_element(By.ID, locator)
.pyenv/lib/python2.7/site-packages/selenium/webdriver/remote/webdriver.py:712: in find_element
  {'using': by, 'value': value})['value']
.pyenv/lib/python2.7/site-packages/selenium/webdriver/remote/webdriver.py:201: in execute
  self.error_handler.check_response(response)
```

Figura 32: Build – Falha em test case

De forma análoga, um projeto foi configurado para o gerente, o qual é responsável pelos *merges* no repositório principal. Uma vez que mais de um desenvolvedor trabalha no mesmo projeto, vários *pull requests* são gerados no repositório central e cabe ao gerente os aprovar ou não. A fim de facilitar e dar confiabilidade a este trabalho, fora então configurado um trabalho para o gerente executar dada uma quantidade *n* de *merges*. Assim, este saberá se alguma funcionalidade do sistema irá quebrar dada a integração dos trabalhos realizados em certo período e pode tomar as devidas providências.

#### 4. DIFICULDADES ENCONTRADAS

Durante a execução das atividades do estágio, algumas dificuldades foram encontradas, principalmente no que diz respeito a falta de informação e documentação. Dado o desconhecimento prévio do tema por parte do autor deste trabalho, muita pesquisa e testes foram necessários durante a execução desse projeto. Os quais foram dificultados principalmente pela falta de documentação em algumas das tecnologias utilizadas.

Apesar da excelente documentação do webdriver utilizado pelo selenium, o qual dispõe de binds e documentações para diversas linguagens, somente a documentação disposta para a linguagem java é excelente. Apesar de existir a documentação para a linguagem Python, diversas vezes o autor teve que recorrer a documentação voltada para o Java.

Além disso, toda a documentação provida ao Selenium Grid é muito escassa e desatualizada. Isto especificamente, ocasionou uma grande perda de tempo.

De forma semelhante, apesar do Jenkins disponibilizar uma excelente documentação, seus plugins deixam muito a desejar. Muito tempo foi gasto testando plugins a fim de selecionar a melhor alternativa para a solução em questão.

Outro ponto bastante difícil foi unir essas tecnologias a ponto de prestar o serviço desejado. Uma vez que cada tecnologia tinha suas limitações, e casos de uso específicos, moldá-las para atender a proposta desse trabalho foi um grande desafio.

Contudo, vários destes problemas vieram a ser facilitados com o auxílio da comunidade open source encontrada no servidor irc freenode, principalmente os canais #selenium, #jenkins e #python, os quais dispõem de centenas de usuários ativos para ajuda em tempo real.

## 5. CONCLUSÕES

Durante o desenvolvimento deste estágio junto a empresa, ficou evidente a importância de fluxos automatizados, principalmente no que diz respeito ao processo de testes e integração de trabalhos.

A possibilidade de um único *test case* poder ser rodado em diferentes plataformas, em diferentes browsers e até versões diferentes destes browsers se notou como algo inovador e surpreendente, visto que mesmo os próprios desenvolvedores apenas testavam o software em um único browser e versão.

Seria ainda interessante alguma forma de medida que provasse a eficácia tanto da qualidade adquirida quanto do tempo economizado ao longo da utilização do framework.

Contudo, dado o problema inicial proposto por este trabalho, o da falta de automação de testes no mercado de trabalho, principalmente de testes em nível de usuário a fim de validar regressivamente os requisitos funcionais de uma aplicação web. Este trabalho cumpre sua função, de forma a inserir no processo de desenvolvimento de software da empresa, o produto aqui demonstrado. Ação esta, que agregou muitas vantagens ao processo de desenvolvimento de software em questão.

Ficou claro tanto para os membros da empresa o quanto para este autor, que este trabalho agregou em qualidade e otimização de recursos no processo de desenvolvimento de software.

## 6. LISTA DE SIGLAS E ABREVIATURAS

|               |  |
|---------------|--|
| Biblioteca    | Uma coleção de software e respectiva documentação, concebida para auxiliar em desenvolvimento de software, uso ou manutenabilidade (“IEEE Standard Glossary of Software Engineering Terminology”, 1990).   |
| Componente    | Uma das partes que compões um sistema. Uma coleção de unidades com interface definida em relação com outros componentes(CRAIG; JASKIEL, 2002).   |
| VCS - SCV     | Version Control System – Sistema de controle de versão.  |
| Driver        | Um módulo de software que invoca, controla e monitora a execução de um ou mais módulos de software (“IEEE Standard Glossary of Software Engineering Terminology”, 1990).   |
| Driver Script | Um script que controla a execução de processos utilizando funcionalidades providas por bibliotecas de teste.   |
| DRY           | Don't Repeat Yourself.   |
| Locator       | Localizador utilizado pelo Selenium a fim de localizar <i>web elements</i> .   |
| Set Up        | Código que é executado antes de cada <i>test case</i> em um <i>test suite</i> . Chamado de pré-condições em testes manuais.  |
| SUT           | System Under Test – Sistema sobre teste.   |
| Tear Down     | Código que é executado depois de cada <i>test case</i> em um <i>test suite</i> .   |
| Test Case     | Um conjunto de entradas, pré-condições e resultados esperados desenvolvidos para um objetivo em específico ou testar condições.  |
| Test Suite    | Uma coleção composta por um ou mais <i>test cases</i> .  |
| TDD           | Test-Driven Development – Desenvolvimento Dirigido a Teste. Técnica de desenvolvimento onde <i>unit tests</i> automatizados são escritas antes mesmo do código de produção. O teste dirige o desenvolvimento do sistema e têm-se como resultado um <i>test suite</i> para regressão. |
| Unit Test     | Testes unitários. Nível de teste para avaliar uma unidade de código. Geralmente são automatizados e escritos pelo programador que escreveu o código a ser testado.   |
| Requisito     | Uma condição que precisa ser tomada por um sistema ou  |

componente para satisfazer um contrato, uma especificação ou outra forma de documento formal imposto. Pode ainda ser funcional ou não funcional.

- Record and play Uma abordagem scriptada onde uma ferramenta de teste grava entradas de teste enquanto utiliza-se a aplicação a ser testada. O resultado pode ser reproduzido depois.
- Web Element Interface pela qual Selenium executa operações sobre um elemento html localizado através de um *locator*.
- Webdriver Interface de controle e introspecção de browsers através de uma API amigável.



## 7. REFERÊNCIAS BIBLIOGRÁFICAS

BURNSTEIN, I. **Practical Software Testing: A Process-Oriented Approach**. [s.l.] Springer Science & Business Media, 2003.

CRAIG, R. D.; JASKIEL, S. P. **Systematic Software Testing**. Boston: Artech House, 2002.

FEWSTER, M.; GRAHAM, D. **Software Test Automation**. Reading, MA: Addison-Wesley Professional, 1999.

FOWLER, M. **bliki: PageObject**. Disponível em:  
<<http://martinfowler.com/bliki/PageObject.html>>. Acesso em: 20 maio. 2015.

FOWLER, M. **Continuous Integration**. Disponível em:  
<<http://martinfowler.com/articles/continuousIntegration.html>>. Acesso em: 20 out. 2015.

GOJARE, S.; JOSHI, R.; GAIGAWARE, D. Analysis and Design of Selenium WebDriver Automation Testing Framework. **Procedia Computer Science**, v. 50, n. 0, p. 341 – 346, 2015.

IEEE Standard Glossary of Software Engineering Terminology. **IEEE Std 610.12-1990**, p. 1–84, dez. 1990.

JOHNSON, R.; FOOT, B. Designing Reusable Classes. **Designing Reusable Classes**, p. 22–35, jun. 1988.

KIT, E. **Integrated, Effective Test Design and Automation**. Disponível em:  
<<http://www.drdoobs.com/integrated-effective-test-design-and-aut/184415652>>. Acesso em: 17 abr. 2015.

LOELIGER, J.; MCCULLOUGH, M. **Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development**. [s.l.] O'Reilly Media, Inc., 2012.

MCKAY, J. **Are there any statistics that show the popularity of Git versus SVN? - Programmers Stack Exchange**. Disponível em:  
<<http://programmers.stackexchange.com/questions/136079/are-there-any-statistics-that-show-the-popularity-of-git-versus-svn>>. Acesso em: 20 maio. 2015.

MITCHELL, R. **Essential browser tools for Web developers**. Disponível em:  
<<http://www.computerworld.com/article/2502388/app-development/essential-browser-tools-for-web-developers.html>>. Acesso em: 20 maio. 2015.

NAGY, D.; YASSIN, A. M.; BHATTACHERJEE, A. Organizational Adoption of Open Source Software: Barriers and Remedies. **Commun. ACM**, v. 53, n. 3, p. 148–151, mar. 2010.

OUSTERHOUT, J. K. Scripting: higher level programming for the 21st Century. **Computer**, v. 31, n. 3, p. 23–30, mar. 1998.

RASHED, G.; AHSAN, R.; OTHERS. Python in computational science: applications and possibilities. **International Journal of Computer Applications**, v. 46, n. 20, p. 26–30, 2012.

ROSSUM, G. V.; DRAKE, F. L. **Python Reference Manual Release 2.2.1**. [s.l.: s.n.].

**Standard glossary of terms used in Software Testing**. Erik van Veenendaal, , set. 2005. Disponível em: <[http://www.cftl.fr/fileadmin/pdf/en/glossary-current\\_v1.1.pdf](http://www.cftl.fr/fileadmin/pdf/en/glossary-current_v1.1.pdf)>

THOLENCE, P. **FirePath**. Disponível em: <<https://addons.mozilla.org/en-us/firefox/addon/firepath/>>. Acesso em: 20 maio. 2015.

TROVALDS, L. **Linus Trovalds on git**: Linus Trovalds on git., 17 abr. 2015. Disponível em: <[https://www.youtube.com/watch?v=UKY3scPIMd8&feature=youtube\\_gdata\\_player](https://www.youtube.com/watch?v=UKY3scPIMd8&feature=youtube_gdata_player)>. Acesso em: 20 maio. 2015.

WANG, H.; WANG, C. Open source software adoption: a status report. **IEEE Software**, v. 18, n. 2, p. 90–95, mar. 2001.

WILSON, G. et al. Best Practices for Scientific Computing. **PLoS Biol**, v. 12, n. 1, p. e1001745, 7 jan. 2014.

## ANEXO I – QUESTIONÁRIO – PERGUNTAS

**Testes de Software**

Este questionário tem a finalidade de adquirir informações para utilização em TCC.

**A empresa em que você trabalha testa o software por ela produzido?\***

- ☐ Sim  
☐ Não  
☐ Não sei

**Sua empresa ou organização realiza testes a nível de usuário?\***

Testes que utilizam ou simulam reais usuários utilizando o sistema.

- ☐ Sim  
☐ Não  
☐ Não sei

**Quem testa o software na sua empresa ou organização?\***

- ☐ Cada um é responsável por testar o que produz.  
☐ Existe uma equipe ou pessoa responsável por testar o software produzido.  
☐ Não sei  
☐ Other:

**Os testes ocorrem a cada iteração do software?\***

Os testes acontecem a cada novo commit, ou versão, etc.

- ☐ Sim  
☐ Não  
☐ Não sei  
☐ Other:

**Sua empresa ou organização utiliza algum sistema de testes automatizado?\***

Testes que são rodados automaticamente, dado algum trigger.

- ☐ Sim  
☐ Não  
☐ Não sei

**Em que empresa ou organização você trabalha?**

Deixe em branco caso não queira informar.

Pergunta 1: A empresa em que você trabalha testa o software por ela produzido?

Pergunta 2: Os testes ocorrem a cada iteração do software?

Pergunta 3: Sua empresa ou organização realiza testes a nível de usuário?

Pergunta 4: Sua empresa ou organização utiliza algum sistema de testes automatizado?

## ANEXO II – QUESTIONÁRIO – RESPOSTAS

|    | Pergunta 1 | Pergunta 2 | Pergunta 3 | Pergunta 4 |
|----|------------|------------|------------|------------|
| 1  | Sim        | Não        | Não        | Não        |
| 2  | Sim        | Sim        | Não        | Não        |
| 3  | Não        | Não        | Não        | Não        |
| 4  | Sim        | Sim        | Não        | Não        |
| 5  | Sim        | Sim        | Não        | Sim        |
| 6  | Sim        | Sim        | Não        | Não        |
| 7  | Sim        | Sim        | Não        | Não        |
| 8  | Sim        | Não        | Não        | Não        |
| 9  | Sim        | Sim        | Não        | Não        |
| 10 | Sim        | Sim        | Não        | Não        |
| 11 | Não        | Não        | Não        | Não        |
| 12 | Sim        | Sim        | Não        | Não        |
| 13 | Sim        | Sim        | Sim        | Não        |
| 14 | Sim        | Sim        | Sim        | Não        |
| 15 | Sim        | Sim        | Sim        | Sim        |
| 16 | Sim        | Sim        | Sim        | Sim        |
| 17 | Sim        | Sim        | Sim        | Não        |
| 18 | Sim        | Sim        | Sim        | Sim        |
| 19 | Sim        | Sim        | Sim        | Não        |
| 20 | Sim        | Sim        | Sim        | Não        |
| 21 | Sim        | Sim        | Sim        | Sim        |
| 22 | Sim        | Sim        | Sim        | Não        |
| 23 | Sim        | Não        | Sim        | Não        |
| 24 | Sim        | Sim        | Sim        | Sim        |
| 25 | Sim        | Sim        | Sim        | Não        |
| 26 | Sim        | Sim        | Sim        | Não        |
| 27 | Sim        | Sim        | Sim        | Não        |
| 28 | Sim        | Sim        | Sim        | Não        |
| 29 | Sim        | Sim        | Sim        | Sim        |
| 30 | Sim        | Sim        | Sim        | Não        |
| 31 | Sim        | Sim        | Sim        | Sim        |
| 32 | Sim        | Sim        | Sim        | Não        |
| 33 | Sim        | Sim        | Sim        | Não        |
| 34 | Sim        | Sim        | Sim        | Não        |
| 35 | Sim        | Sim        | Sim        | Sim        |
| 36 | Sim        | Sim        | Sim        | Sim        |
| 37 | Sim        | Sim        | Sim        | Sim        |
| 38 | Sim        | Sim        | Sim        | Não        |
| 39 | Sim        | Não        | Sim        | Não        |
| 40 | Sim        | Não        | Sim        | Não        |
| 41 | Sim        | Sim        | Sim        | Não        |
| 42 | Sim        | Sim        | Sim        | Sim        |
| 43 | Sim        | Sim        | Sim        | Não        |
| 44 | Sim        | Não        | Sim        | Não        |
|    | 95.45%     | 81.82%     | 72.72%     | 27.28%     |
|    | 4.55%      | 18.18%     | 27.28%     | 72.72%     |