ETA - Easy Test Automation: uma ferramenta para automação de testes funcionais web baseada em Selenium Webdriver e TestNG

Rafael Amaral de Santana
Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA
Salvador - Bahia - Brasil
rafaelamaral@ifba.edu.br

Flávia Maristela Santos Nascimento
Instituto Federal de Educação, Ciência e Tecnologia da Bahia - IFBA
Salvador - Bahia - Brasil
flaviamsn@ifba.edu.br

RESUMO

O presente trabalho apresenta uma ferramenta desenvolvida em Java, para realização de testes funcionais em aplicações *web*, o ETA (*Easy Test Automation*). Com a utilização do ETA é possível realizar a separação entre o SUT (*System Under Testing*), os *scripts* e os dados que serão utilizados nos testes. A abordagem utilizada pelo ETA na realização dos testes automatizados diminui a curva de aprendizado, aumenta a flexibilidade e facilita a criação e manutenção dos *scripts* de testes automatizados.

Palavras Chave

Teste de software, Automação de testes, Selenium, TestNG, Aplicações Web, Testes funcionais

1. INTRODUÇÃO

A crescente presença dos *softwares* no cotidiano das pessoas tem provocado um aumento perceptível nos níveis de interação entre usuários e os diversos tipos de dispositivos computacionais, desde computadores pessoais passando por *notebooks* e *smartphones* e chegando até dispositivos mais modernos, como por exemplo os óculos interativos [12]. Diante dessa nova realidade, surge o desafio de entregar sistemas com cada vez mais qualidade para os nossos usuários.

Com a adoção de práticas ágeis, as empresas tem buscado aumentar a produtividade, reduzindo o custo e o esforço na construção de *softwares*, o que pode acarretar em uma baixa qualidade do produto final. Preocupações com a qualidade do código e da implementação muitas vezes são ignoradas ou deixadas para as fases finais, que com a grande pressão pela entrega do produto, acabam não recebendo a devida atenção [14]. Diante disso, técnicas para integração contínua de código e automatização dos processos de desenvolvimento de *software* têm surgido frequentemente, com o intuito de atribuir maior confiança e qualidade ao desenvolvimento ágil de *software*. Dentro desse cenário, a automação de testes de *software* tem se destacado.

A intenção deste trabalho é apresentar o *Easy Test Automation* - ETA, uma proposta de solução para testes automatizados em aplicações *web*, cujos requisitos foram:

- Executar testes automatizados em aplicações *web*;
- Gerar evidências das execuções dos testes;
- Disponibilizar relatório contendo informações da execução dos testes;
- Permitir a criação de casos de teste de maneira simplificada;
- Desacoplar os casos de teste da massa de dados utilizada nos testes.

O ETA tem como função principal reduzir o esforço do testador para automatizar a execução dos testes no SUT (*System Under Testing*). Isso pode ser alcançado através da abordagem utilizada, onde um desenvolvedor será responsável por fornecer ao testador os métodos para interação com SUT. Dessa forma, o testador pode abstrair a forma como o ETA realiza a automação de testes, e concentrar-se na elaboração dos *scripts* XML, podendo assim ter uma baixa curva de aprendizado para realização de testes automatizados em aplicações *Web*. O ETA permite gerar relatórios em português[1] de forma a facilitar o entendimento dos *stakeholders*. Para construção do ETA foram utilizadas boas práticas, padrões de projetos Java e de automação de testes [21] [18].

2. ESTRUTURA DO TRABALHO

Este documento está organizado como segue. A Seção 3 trata da justificativa, motivação e objetivo dessa proposta. A Seção 4 apresenta as ferramentas e trabalhos relacionados ao ETA. A Seção 5 apresenta uma breve fundamentação teórica sobre Qualidade de *Software* e automação da execução de testes. Na Seção 6 são apresentadas as técnicas e padrões utilizados na construção do projeto. Na Seção 7, Seção 8 e Seção 9 serão abordados os conceitos referentes construção do ETA, sua estrutura de pacotes e classes, tecnologias, ambiente de execução e ferramentas auxiliares que foram utilizadas para desenvolvimento dessa abordagem. Na Seção 10 é apresentado o método utilizado para realizar os testes e validações da aplicação. Nas Seção 11 será apresentada a conclusão do trabalho e na Seção 12 serão descritas as possibilidades de novas funcionalidades e melhorias a serem feitas na ferramenta.

3. MOTIVAÇÃO E JUSTIFICATIVA

Com o passar dos anos, as empresas começaram a compreender a importância de entregar produtos de qualidade para os seus usuários. A história apresenta uma série de exemplos, em que *softwares* que não funcionaram como esperado,

[1]O relatório gerado pela biblioteca Extent Reports [13] foi traduzido para português pelo autor do ETA. A tradução foi submetida aos autores da biblioteca, mas ainda não está disponível para a comunidade.

trouxeram prejuízos financeiros, materiais e, em alguns casos, ameaças a integridade física das pessoas [27]. Com isso, a prática de testar os *softwares* vem crescendo e se tornando comum dentro do processo de desenvolvimento de *software*.

Em um processo de desenvolvimento de *software* maduro[2], a necessidade de se automatizar os testes surge naturalmente. A incorporação da automação de testes ao projeto, pode trazer diversos benefícios a médio e longo prazo, como por exemplo, diminuição da quantidade de erros decorrentes da implantação de novos módulos no sistema, geração de evidências de forma mais agilizada e *feedback* mais rápido sobre a qualidade do que está sendo desenvolvido. Dessa forma é possível reduzir os custos e o tempo gasto na execução de testes e aumentar a produtividade, sem afetar a qualidade do produto final.

Hoje, a automação de testes de *software* é uma prática muito valorizada e desejada dentro de um processo de desenvolvimento de *software*. Existem algumas razões para que a equipe decida por automatizar a execução dos testes, dentre elas podemos citar:

- Reduzir a probabilidade de ocorrem erros dos testadores nos testes manuais;
- Reduzir o tempo de execução dos testes, já que em geral testes manuais levam mais tempo para serem executados;
- Reduzir o esforço na realização de tarefas repetitivas, como por exemplo, o preenchimento de um mesmo formulário diversas vezes com dados diferentes;
- Reutilizar os testes em sistemas que terão um longo ciclo de vida/desenvolvimento, ou seja, que serão utilizados por anos ou o seu desenvolvimento irá conter diversos ciclos de iteração;
- Permitir que testadores se preocupem com atividades criativas, como execução de testes exploratórios e *design* de casos de teste;
- Auxiliar nos testes de regressão dos sistemas.

Estes aspectos, quando levados em consideração e aplicados corretamente junto com as diretrizes e orientações para automação, podem proporcionar um bom retorno sobre o investimento[3] dos testes automatizados. Considerando a redução de tempo para a execução de testes manuais e automáticos como o principal beneficio associado a utilização da automação no projeto [17].

É possível afirmar que na implantação de um processo de testes automático haverá uma necessidade maior de investimentos em tempo e recursos (humanos e financeiros) do que na implantação de testes manuais. Isto porque, em geral é necessário realizar a aquisição de ferramentas, treinamentos, provas de conceito e até contratação de novos profissionais especializados. Essas são algumas justificativas para o aumento dos investimentos em recursos humanos e financeiros. Entretanto, a expectativa é que este investimento seja retornado no decorrer do projeto.

A partir de uma simples pesquisa de mercado foi possível identificar uma lacuna no que diz respeito a uma solução gratuita que agregasse valor aos testes do produto final, antes da entrega ao cliente. Assim, a ideia para concepção de uma ferramenta que pudesse:

1. Prover uma solução com um bom custo benefício e baixa curva de aprendizado;
2. Aumentar produtividade dos testadores, e minimizar as interações manuais realizadas durante a execução dos testes;
3. Diminuir o custo de manutenção;
4. Ser de fácil configuração.

Pensando nas necessidades das equipes que buscam a automação de testes e a lacuna encontrada no mercado. O ETA se propõe a trazer alguns benefícios associados a sua utilização, tais como:

- Redução do risco de novas falhas em áreas estáveis do SUT, resultantes da inserção de novas funcionalidades;
- Facilidade de execução de testes do SUT em diferentes ambientes (Sistemas operacionais, *Browsers* e dispositivos);
- Facilidade de manutenção dos casos de teste automatizados (*scripts*);
- Facilidade de entendimento dos *stakeholders*, através da apresentação de relatórios gerenciais e técnicos;
- Redução do custo de manutenção do SUT, pois permite a realização de testes de regressão com maior agilidade.

Visando propiciar esses benefícios e satisfazer os seus requisitos, o ETA tem como principais objetivos:

- Ser uma ferramenta de código aberto que possa ser mantida e melhorada pela comunidade;
- Facilitar a implantação da automação de testes dentro do processo de desenvolvimento;
- Facilitar o entendimento do testador/programador responsável pela criação dos *scripts* de teste;
- Permitir a execução de testes diferentes plataformas e nos mais diversos *browsers* disponíveis no mercado;
- Gerar evidências de sucesso e falha dos testes através de *logs*, *screenshots* e *screencasts*;
- Gerar relatórios detalhados que agreguem valor aos *stakeholders* do SUT;
- Permitir utilização de *Data Driven* para gestão de massa de dados dos testes;
- Prover execuções de suítes de testes, execução de testes em paralelo e ordenadamente.

Na Seção 4 serão descritas as ferramentas relacionadas a ferramenta proposta.

[2]Um processo de desenvolvimento é tido como maduro, quando há uma consistência na forma como o *software* é produzido, definido, documentado e constantemente melhorado. O desenvolvimento dos projetos é visível, a utilização do processo é controlada e medida, a inserção de novas tecnologias e melhorias no processo é feita de forma disciplinada [19].

[3]ROI - *Return of Investment*

4. TRABALHOS RELACIONADOS

Na última década, a automação de testes ganhou destaque e vem se tornando objeto de desejo das empresas que desenvolvem *softwares*. Por outro lado, por ser uma área imatura (comparada a outras áreas da engenharia de *software* [27]), pode não representar um bom negócio, caso a implementação seja feita sem alinhamento das expectativas e planejamento.

Nesse período, muitas lições foram aprendidas pelos fornecedores de ferramentas de automação e clientes, e o resultado disso é que hoje as ferramentas estão ficando cada vez mais robustas e fáceis de manusear.

Há pelo menos uma geração, as ferramentas de testes começaram a se tornar mais acessíveis para usuários não programadores. Ainda assim, apesar das tentativas, as ferramentas ainda dependiam da gravação de *scripts*, uma tarefa complexa e inflexível, em que normalmente, para realizar manutenções nesses *scripts* era necessário conhecimentos em linguagens de programação, ou seja, o testador deveria em alguns momentos atuar como programador.

Por isso, antes de escolher uma ferramenta de automação de testes, é importante avaliar quais as reais necessidades do projeto e dos *stakeholders* e analisar se as ferramentas disponíveis no mercado atendem esses requisitos. Uma ferramenta por mais recursos que ofereça, pode não atender as necessidades básicas do projeto e por isso é necessário analisar a viabilidade de adoção.

Nessa Seção são descritas as principais ferramentas relacionadas ao ETA. Os pontos fortes e fracos de cada abordagem são sumarizados, ao final da seção, na Tabela 1, será apresentado um comparativo entre as soluções.

A maioria das ferramentas apresentadas estão sob a licença *open source*, todas tem como finalidade automatizar o processo de execução dos testes de aceitação e funcional, facilitam a implantação da automação, apresentam facilidades na criação de *scripts* e geram relatórios e evidências dos testes.

4.1 Serenity BDD

O Serenity é uma biblioteca de código aberto para BDD[4], que ajuda a escrever testes de aceitação e de regressão com maior agilidade, mais simples de compreender e com menor esforço para realizar a manutenção [9].

A biblioteca também apresenta os resultados de forma a ilustrar e facilitar o entendimento e descrição da suíte de testes que foi executada. Além disto, é possível, saber não apenas o que foi executado, mas também quais requisitos estão cobertos por esses testes.

Uma das principais vantagens de usar Serenity BDD é que, não é necessário investir tempo na construção e manutenção de uma solução estruturada para automação uma vez que, essa ferramenta possui as principais funcionalidades necessárias a um projeto típico de automação de testes.

Serenity fornece apoio nos testes automatizados *web* usando o Selenium WebDriver [8], embora também funcione de forma eficaz para testes não-*web*, tais como testes que executam serviços *web* (REST) ou até mesmo chamando o código do aplicativo diretamente.

O objetivo da Serenity é tornar fácil a escrita de testes de aceitação, que possam ser facilmente atualizados e automatizados, usando uma biblioteca de testes BDD ou convencional. É possível trabalhar com ferramentas BDD como Cucumber[5] ou JBehave,[6] ou simplesmente usar o JUnit [15].

É possível realizar integração com os requisitos armazenados em um servidor externo ou qualquer outra ferramenta de gerenciamento de casos de teste, ou simplesmente usar uma abordagem baseada em diretório simples.

Diante de suas características, o Serenity se apresenta como uma boa alternativa para testes de aceitação de usuário, é uma ferramenta de código aberto. Entretanto, é necessário pagar para obter treinamento e suporte especializado. Ao optar pela utilização da ferramenta, é possível escrever os testes através de *scripts* em alto nível em inglês.

4.2 IBM Rational® Functional Tester

O *Rational® Functional Tester* - RFT é um dos *softwares* que compõem a família de produtos *Rational* da empresa IBM, é um *framework* utilizado para realizar teste automatizados funcionais e de regressão em aplicações Java *Desktop*, *web*, .NET baseadas em terminal, SAP, Siebel e *web*2.0. Segundo o *site* da fabricante as principais funcionalidades disponíveis no RFT são [3]:

- Permite o uso de *Keywords* (palavras chave);
- Permite a criação de *scripts* em diversas linguagens de programação, dentre elas Java, Microsoft .NET e Visual Basic;
- Permite validar dados dinâmicos com diversos pontos de verificação e apoio a expressões regulares;
- Minimiza o tempo gasto e reduz o retrabalho na criação dos *scripts*;
- Apoia o controle de versões, o que permite o desenvolvimento paralelo de *scripts* e permite o uso por equipes geograficamente distantes;
- Possui uma abordagem diferenciada para utilização de *Data-Driven*.

Por ser uma ferramenta paga, não é possível encontrar muitos dados sobre sua utilização e, em geral só estão disponíveis informações comerciais.

4.3 Marathon

O Marathon é um *framework* para automação de testes, que possui duas versões disponíveis para utilização, o *MarathonITE* (*Marathon Integrated Testing Environment*), e o *Marathon*. O ITE é a versão paga da ferramenta enquanto a outra é distribuída sob a licença *open source*.

Ambos são aplicações que têm como finalidade a automação de testes para aplicações Java/Swing [4]. Eles fornecem suporte para linguagens de *script* como Ruby e Python e permitem acesso direto à aplicação dos *scripts* de teste. Usando o Marathon é possível gravar e executar testes, gerar relatórios e integrar com ferramentas externas para estender as funcionalidades da aplicação.

Entre as principais funcionalidades disponíveis na versão básica (*open source*) estão:

[4] *Behavior Driven Development* - Desenvolvimento Guiado por Comportamento

[5] O Cucumber é um *framework* para escrita de especificações executáveis. Inicialmente foi desenvolvido para Ruby, mas depois ganhou suporte a dezenas de outras linguagens de programação.

[6] O JBehave é um *framework* para escrita de especificações executáveis, nativo do Java.

- Gravação das interações do usuário com o sistema alvo do teste para posterior reprodução dos passos realizados;

- Suporte a testes exploratórios, é possível realizar *screenshots* e criar anotações de resultados;

- Geração de relatório com todas as informações da execução, além disso, é possível exportar os *scripts* para auxiliar na reprodução de erros e futuros testes.

O Marathon na sua versão gratuita é de fácil instalação e utilização, é indicado para testes funcionais. Sua abordagem *record and replay*, permite criar *scripts* de maneira rápida, apenas gravando a utilização do usuário na aplicação alvo dos testes.

Dessa forma não é necessário nenhum conhecimento técnico em linguagem de programação, caso seja necessário alterar os *scripts* a ferramenta permite a exportação desses testes gravados para Java ou Ruby.

5. VALIDAÇÃO, VERIFICAÇÃO E TESTES

O processo conhecido como "VV&T" é composto por três atividades: Validação, Verificação e Teste [12]. A validação tem por objetivo assegurar que o produto final da construção atenda aos requisitos que foram definidos. Neste ponto, a pergunta a ser respondida é: "Estamos construindo o produto certo?".

A verificação tem como responsabilidade garantir que o *software* está sendo construído da maneira correta em termos de técnicas, tecnologias e processos utilizados. A pergunta a ser respondida é: "Estamos construindo o produto corretamente?".

A terceira atividade do "VV&T" é o teste do *software*, cujo objetivo é analisar o *software* construído, examinando o seu comportamento através de sua execução ou inspeção do código, visando assegurar que o que foi construído funciona corretamente. A pergunta a ser respondida nesse momento é, "O produto construído funciona corretamente?" [20].

O VV&T abrange uma série de atividades relacionadas a garantia da qualidade do *software*. Essas atividades podem ser divididas em dois grandes grupos, de acordo com as suas características [14]:

- Atividades Estáticas: Utilizam métodos que não envolvam a execução do produto ou de um modelo para determinar e/ou estimar a qualidade do *software*. Exemplos de métodos estáticos seriam as revisões formais (inspeção[7], *Walktroughs*[8], entre outros).

- Atividades Dinâmicas: São aquelas que envolvem a execução do *software* com dados, em ambiente real (produção) ou simulado. Exemplos de atividades dinâmicas são Automação de Testes, Teste unitários e Testes de mesa[9].

5.1 O Teste de Software

O teste de *software* não é uma atividade trivial: requer cautela e atenção. Além disso, os seres humanos são passíveis de erros, que podem ser causados por uma infinidade de razões, sejam elas físicas, emocionais, ambientais, dentre outras [14].

Em resumo, o teste do *software* é a investigação do *software* a fim de fornecer informações sobre sua qualidade em relação ao contexto em que ele deve operar. Isso inclui o processo de utilizar o produto para encontrar seus defeitos.

Não se pode garantir que todo *software* funcione corretamente, sem a presença de erros, visto que os mesmos, muitas vezes, possuem um grande número de estados com fórmulas, atividades e algoritmos complexos [14].

A atividade de construção de *software* pode ser complexa dependendo das características, do tamanho e da quantidade de pessoas envolvidas no processo.

Idealmente, toda permutação possível do *software* deveria ser testada. Entretanto, isso se torna impossível para a ampla maioria dos casos, devido à quantidade impraticável de possibilidades. A qualidade do teste acaba se relacionando à qualidade dos profissionais envolvidos em filtrar as permutações relevantes e as técnicas utilizadas para realizar para realizar o teste do *software* [26].

Fatores humanos, ambientais, estruturais e de outras naturezas, podem colaborar para a ocorrência de erros e que o produto final não esteja em conformidade com o que foi especificado no início do projeto [20].

Com a popularização da *Internet* e crescente utilização de sistemas computacionais no nosso dia a dia, o risco de acontecerem problemas causados por um *software* que não funciona como se esperava, aumenta à medida que sua utilização se torna mais frequente.

As consequências desses problemas podem levar a prejuízos financeiros, a reputação de uma companhia e até ameaçar a integridade da pessoas. Por esses e outros motivos é necessário realizar testes no *software*. O ser humano em sua natureza é suscetível a erros, por isso, é muito provável que atividades que envolvam ações humanas possuam desvios e erros causados por essa interação [12].

Por outro lado, apenas testar o *software* deliberadamente não é suficiente. Apenas os testes não podem garantir que todos os erros tenham sido encontrados[10], idealmente diversas técnicas devem ser combinadas para ampliar a cobertura dos testes e descobrir a maior quantidade de *bugs* possíveis. Aliados a processos de testes bem definidos, as técnicas de VV&T podem garantir uma maior cobertura do produto [26] [27].

Conceitualmente, os Testes são divididos em duas técnicas (Caixa Branca e Caixa Preta), distribuídos em cinco níveis (Teste de Unidade, de Integração, de Sistema, de Aceitação e Integrado) e quatro tipos (Teste Funcional, Não-Funcional, Estrutural e de Manutenção) [14] [12] [26]. O teste é um

[7]Inspeção formal, cujo os papéis dos participantes são bem definidos, possuem critérios de entrada e saída definidos previamente e seguem um processo formal baseado em regras e na utilização de *check-list*. A inspeção é conduzida por um moderador que não pode ser o autor, utilizando métricas para realização da reunião.

[8]O *walktrought*, ou acompanhamento, é um tipo de revisão conduzida pelo autor, que tem caráter informativo e de aprendizagem, pode ser informal ou formal, geralmente é utilizada para disseminar determinado conhecimento sobre o produto.

[9]Tipo de teste que simula a execução do algoritmo sem um computador (compilador ou interpretador), sendo realizado com o auxilio de papel e caneta

[10]O 2º Principio do teste diz que: Teste exaustivo (100% de cobertura de combinações de entrada e pré-condições) é impossível, exceto em casos triviais [14].

Tabela 1: Ferramentas Relacionadas

Características/ Ferramentas	**ETA - *Easy Test Automation***	**Serenity BDD**	**RFT - *Rational Functional Test***	**Marathon ITE - *Integrated Testing Environment***	**Marathon**
Licença	*Open Source*	*Apache 2 Open Source*	Proprietária	Proprietária	*LGPL Open Source*
Nível de teste	Funcional	Aceitação	Funcional	Aceitação/ Funcional	Funcional
Tipo de plataforma	*Web*	*Web/ Desktop*	*Web*	*Web*	*Web*
Conhecimentos necessários para criação de scripts	XML	Inglês/ Cucumber ou JBehave	Java e RFT	Java ou Ruby	Java ou Ruby
Relatórios de Execução	Sim	Sim	Sim	Sim	Sim
Relatórios detalhados de execução	Sim	Sim	Sim	Sim	Não
Idioma dos relatórios	Português/ Inglês/ Espanhol	Inglês	Inglês	Inglês	Inglês
Evidências	*Screenshots/ Screencasts/ Logs de execução*	*Screenshots*	*Screenshots*	*Screenshots*	*Screenshots*
Data Driven	Sim	Sim	Sim	Sim	Não
Page Objects	Sim	Sim	Sim	Não	Não
Object Map	Não	Não	Sim	Sim	Não
Cross Browser	Sim	Sim	Sim	Sim	Não
Cross Plataform	Sim	Sim	Sim	Sim	Sim
Execução Paralela	Sim	Não	Não	Sim	Não
Teste Mobile	Não	Não	Não	Não	Não
Formato de *scripts*	XML	Linguagem Natural (Inglês)	Java	Java ou Ruby	Java ou Ruby

processo realizado pelo testador de *software*, que permeia outros processos da engenharia de software, e que envolve ações que vão do levantamento de requisitos até a execução do teste propriamente dito [25].

5.2 O Processo de Testes

> "Um bom testador, requer um bom entendimento do processo de desenvolvimento e do produto que está sendo gerado, além da habilidade de indicar possíveis falhas e erros. Ele também deve ter uma atitude de questionar todos os aspectos relacionados com o *software*".

As atividades de verificação e validação estão distribuídas nas diversas etapas do processo de testes e, de uma maneira geral, a verificação dos artefatos e do *software* é realizada antes da validação. Por isso, existem técnicas, níveis e tipos de teste cada um com seus objetivos distintos e visando contribuir de maneiras diferentes para a garantia de qualidade do produto final [20].

Um processo de teste relaciona um conjunto de etapas, atividades, artefatos, papéis e responsabilidades que juntos têm como finalidade estruturar e padronizar um projeto de testes [25]. Segundo o *Capability Maturity Model Integration* - CMMI, "A qualidade de um sistema ou produto é amplamente influenciada pela qualidade do processo utilizado" [19], logo a qualidade do modelo (processo) de produção, utilizado para a construção de um *software*, influi diretamente na qualidade do produto gerado.

5.3 Papéis e Responsabilidades de teste

As atividades aqui descritas contextualizam, de uma maneira geral, as responsabilidades de cada cargo dentro do processo de testes, nem sempre elas são seguidas a risca e muitas vezes um analista pode estar paralelamente atuando como Gestor e/ou Testador, a depender das necessidades e demandas do projeto, por exemplo [12].

Gestor de teste

O Gestor de Teste tem por função primordial o planejamento e acompanhamento do projeto de teste do *software*. Ele deve entender e identificar os requisitos de teste definidos para o *software*, definindo a estratégia de testes que será adotada. Entre suas responsabilidades destacam-se a gestão das atividades de especificação e realização dos testes e a consolidação dos resultados do teste.

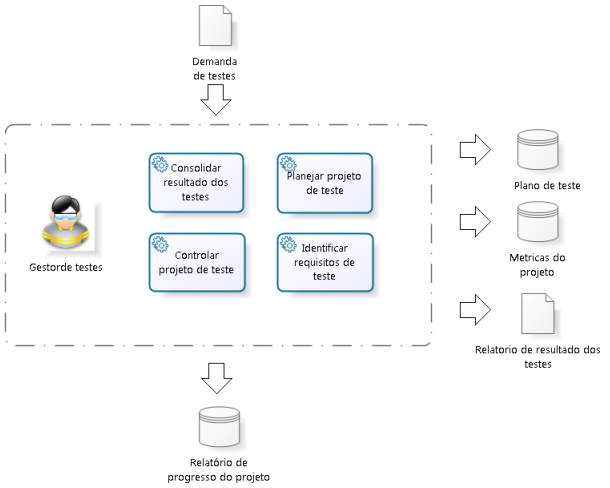

Figura 1: Atividades do gestor de testes

O gestor deve possuir experiência no planejamento de testes, bem como, conhecimento sobre a implementação dos testes. Deve ainda possuir conhecimentos sobre redes de comunicação, modelos de projeto e de bancos de dados, especificação e otimização de performance e automação de testes.

A Figura 1 apresenta um exemplo dos artefatos do gestor de testes. Como entrada ele recebe a demanda de testes e pode gerar como saída o plano de teste, as métricas do projeto e os relatórios de progresso e resultados dos testes.

Analista de teste

O Analista de Testes tem por função a implementação e a avaliação dos testes a serem realizados na aplicação. Entre suas responsabilidades destaca-se a elaboração dos casos de teste, dos procedimentos de teste e dos *scripts* de teste. O Analista de Testes deve preparar o teste a ser executado pelo Testador.

O perfil do analista ou projetista de testes deve corresponder ao profissional com experiência na implementação e na execução de testes manuais e automatizados. Seus principais conhecimentos devem ser nas linguagens de programação das ferramentas de automação de testes, técnicas de testes manuais e tecnologias utilizadas para especificação e automação de testes.

A Figura 2 apresenta as atividades do analista de testes. É possível visualizar os artefatos de entrada e saída, além das atividades realizadas pelos analistas de teste. A depender da senioridade do analista, ele

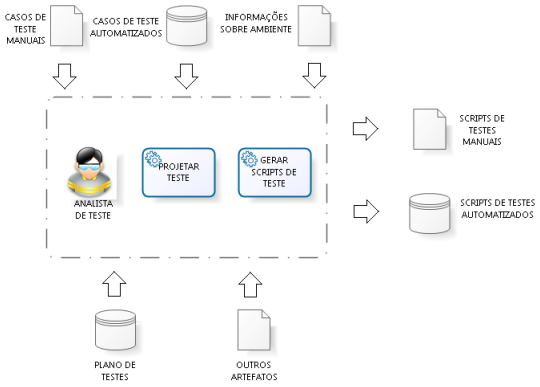

Figura 2: Atividades do analista de testes

pode receber os casos de teste e produzir os *scripts* ou a partir do plano de teste produzir os casos de teste.

Testador

O Testador corresponde ao nível inicial na hierarquia da área de testes, sua função principal é executar os testes elaborados no projeto. Ele deve executar os testes definidos pelo Analista de Testes, avaliar a execução e os resultados dos testes realizados e registrar os erros (incidentes) encontrados, identificando o tipo e a causa da falha. Além disso, ele é o responsável por criar os *scripts* de testes automatizados com base no que foi especificado pelo analista de testes.

A Figura 3 apresenta as atividades e artefatos processados pelos testadores.

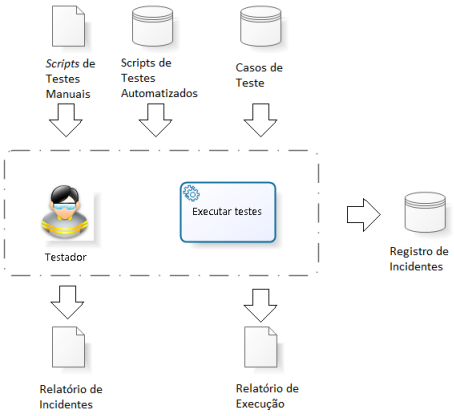

Figura 3: Atividades do testador

Automatizador

A função de automatizador pode ser desempenhada por um analista de testes com experiência em programação ou por um programador. O principal objetivo dessa função é codificar e preparar o ambiente do *framework* de automação de testes que será utilizado pelos testadores do projeto. Seus principais conhecimentos devem ser na linguagem de programação utilizada para desenvolvimento do *framework*, conceitos básicos, boas práticas e técnicas de automação de testes.

A Figura 4 apresenta as tarefas desempenhadas pelo automatizador de testes, além da configuração do ambiente o automatizador pode realizar a execução dos testes automatizados.

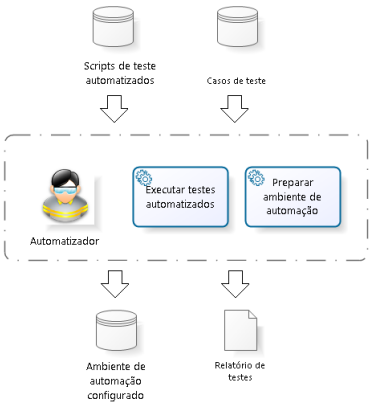

Figura 4: Atividades do automatizador

5.4 Níveis de teste

Teste de unidade

Também conhecida como teste unitário ou teste de módulo, é a fase em que se testam as menores unidades de *software* desenvolvidas (pequenas partes ou unidades do sistema). O alvo desse tipo de teste são as subrotinas, métodos, classes ou mesmo pequenos trechos de código. Assim, o objetivo é encontrar falhas dentro de uma pequena parte do sistema que esteja funcionando, independentemente das demais.

Teste de integração

Na fase de teste de integração, o objetivo é encontrar falhas provenientes da integração interna dos componentes de um sistema. Geralmente os tipos de falhas encontradas são de transmissão de dados. Por exemplo, um componente A pode estar aguardando o retorno de um valor X ao executar um método do componente B; porém, B pode retornar um valor Y, gerando uma falha. O teste de integração conduz ao descobrimento de possíveis falhas associadas à *interface* do sistema. Não faz parte do escopo dessa fase de teste o tratamento de *interfaces* com outros sistemas (integração entre sistemas). Essas *interfaces* são testadas na fase de teste de sistema, apesar de, a critério do gerente de projeto, estas *interfaces* podem ser testadas mesmo antes de o sistema estar plenamente construído.

Teste de validação (Integrado)

Nessa fase, o teste é conduzido pelos administradores do ambiente final em que o sistema ou *software* entrará em ambiente produtivo. Vale ressaltar que essa fase é aplicável somente a sistemas de informação próprios de uma organização, cujo acesso pode ser feito interna ou externamente a essa organização. Nessa fase de teste devem ser feitas simulações para garantir que a entrada em produção do sistema será bem sucedida. Envolve testes de instalação, simulações com cópia de segurança dos bancos de dados, etc.. Em alguns casos um sistema entrará em produção para substituir outro e é necessário garantir que o novo sistema continuará garantindo o suporte ao negócio.

Teste de sistema

Na fase de teste de sistema, o objetivo é executar o sistema sob ponto de vista de seu usuário final, varrendo as funcionalidades em busca de falhas em relação aos objetivos originais. Os testes são executados em condições similares – de ambiente (*hardware* e *software*), *interfaces* sistêmicas e massas de dados – àquelas que um usuário utilizará no seu dia-a-dia de manipulação do sistema. De acordo com a política de uma organização, podem ser utilizadas condições reais de ambiente (*hardware* e *software*), *interfaces* sistêmicas e massas de dados.

Teste de aceitação

Geralmente, os testes de aceitação são realizados por um grupo restrito de usuários finais do sistema, que simulam operações de rotina do sistema de modo a verificar se seu comportamento está de acordo com o solicitado. Teste formal conduzido para determinar se um sistema satisfaz ou não seus critérios de aceitação e para permitir ao cliente determinar se aceita ou não o sistema. Validação de um *software* pelo comprador, pelo usuário ou por terceira parte, com o uso de dados ou cenários especificados ou reais. Pode incluir testes funcionais, de configuração, de recuperação de falhas, de segurança e de desempenho.

5.5 Tipos de teste

Teste Funcional

O objetivo do teste funcional é validar se o sistema atende aos requisitos definidos. Testar "o que" o sistema faz se as suas funcionalidades estão de acordo com o que foi definido em caso de uso, especificações de requisito ou qualquer outro artefato que documente o sistema.

Teste Não Funcional

O objetivo desse tipo de teste, é validar os requisitos não funcionais estão implementados corretamente. Exemplos de testes não funcionais são: testes de carga, performance, usabilidade e instalação.

Teste Estrutural

O teste estrutural visa verificar como o sistema foi construído. Nesse tipo de teste, o código da aplicação pode ser avaliado, pode também verificar a estrutura interna do programa com o intuito de atestar que estão sendo utilizadas boas práticas de programação, padrões de projeto e se há código que nunca é executado.

Teste de Manutenção

O teste de manutenção pode ser de dois tipos:

- Confirmação - É o teste de uma ocorrência após sua correção. Com isso o testador atesta que a ocorrência aberta anteriormente foi corrigida com sucesso;

- Regressão - Teste realizado após uma manutenção ou mudança no sistema (por exemplo: inclusão de novos módulos). O objetivo desse teste é garantir que as funcionalidades que já haviam sido testadas anteriormente, não sofreram impacto após a realização das mudanças.

5.6 Técnicas de teste

Caixa branca

Também chamado de teste estrutural ou orientado à lógica, a técnica de caixa-branca avalia o comportamento interno do componente de *software*. Essa técnica trabalha diretamente sobre o código fonte do componente de *software* para avaliar aspectos tais como: teste de condição, teste de fluxo de dados, teste de ciclos, teste de caminhos lógicos, códigos nunca executados.

Caixa preta

Também chamada de teste funcional, teste comportamental, orientado a dado ou orientado a entrada e saída, a técnica de caixa-preta avalia o comportamento externo do componente de *software*, sem se considerar o comportamento interno do mesmo. Dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado esperado previamente conhecido. Como detalhes de implementação não são considerados, os casos de teste são todos derivados da especificação.

5.7 Testes Manuais

Teste manual é o processo de teste de *software*, utilizado para encontrar defeitos manualmente. Ele requer um testador para desempenhar o papel de um usuário final e usar a maioria ou todos os recursos do aplicativo para garantir o comportamento correto de acordo com as especificações do sistema.

Para garantir a integridade do teste, o testador muitas vezes segue um plano de teste escrito pelo Analista de Testes que os leva através de um conjunto de casos de teste importantes.

5.8 Testes Automatizados

A automação surge para muitas empresas e gestores como a solução ideal para a resolução de todos os problemas de qualidade do *software* que eles estão desenvolvendo. Que ao ser incorporada ao projeto, irá resolver todos os problemas do processo e de qualidade do *software*. Na prática não é bem assim, a necessidade de automatizar a execução dos testes surge naturalmente dentro de um processo bem definido e maduro de desenvolvimento de *software* [25].

Aliado a outras técnicas e processos, a automação pode trazer muitos benefícios e incorporar mais qualidade ao produto desenvolvido. Entretanto, as expectativas precisam ser bem alinhadas entre todos os envolvidos no projeto, expectativas irreais, concepções erradas e até ânsia por retorno rápido pode frustrar e desacreditar o processo de automação.

Alguns erros são comuns e acontecem com certa frequência. Uma das principais causas do fracasso da implantação de um projeto de automação de testes de software está no fato de que as empresas muitas vezes estão desesperadas atrás de resultados que iniciam o projeto sem nenhuma estrutura ou suporte. Adquirem uma ferramenta de automação prematuramente, sem ter o mínimo de qualidade e maturidade no seu processo de teste.

Este processo é informal, os papéis não são bem definidos, os profissionais não possuem perfil adequado ou muitas vezes não são qualificados e a equipe de testes não tem autonomia ou liberdade para opinar sobre o processo de construção do software. A qualidade do produto final é diretamente proporcional a qualidade do processo utilizado no seu ciclo de vida, a automação dos testes requer testes manuais consistentes e maduros, é impossível se alcançar o sucesso do projeto de automação baseando-se num processo de testes manual falho, imaturo e desorganizado [22] [25] [19].

A automação deve ser considerada como um projeto com características e ciclo de vida próprios que será incorporado ao ciclo de vida do desenvolvimento do produto principal, o fruto desse projeto são os *scripts* e ou ferramentas que auxiliarão no processo de automação.

Como projeto a automação de testes exige planejamento detalhado, possui ainda atividades relacionados ao projeto, requisitos, desenvolvimento e testes. Além da necessidade de pessoas qualificadas e aptas a atuar com o projeto, em geral um analista de testes ou testador com experiência em programação é denominado "automatizador" e será responsável pela criação dos *scripts*.

Com base na experiência de mercado, alguns aspectos que devem ser considerados e que foram encontrados em bons projetos de automação de teste de *software*, estão listados a seguir:

- A automação deve ser considerada como um projeto com características, riscos, custos e requisitos próprios;
- Como todo processo que envolve interação humana os *scripts* e as ferramentas de automação podem apresentar defeitos. Esses riscos deve ser conhecidos e devem ser tomadas medidas para mitigar os impactos;
- Devem ser aplicadas as melhores práticas de desenvolvimento de software na construção dos *scripts* e na ferramenta(caso não seja adquirida uma ferramenta pronta);
- Os participantes do projeto de automação devem demonstrar interesse pelo projeto, conhecimento e perfil para atuar tanto com testes, quanto com desenvolvimento;
- A equipe deve ter acesso a hardware adequado (que apresente bom desempenho) as caraterísticas do projeto de automação, da ferramenta e do ambiente onde ela será executada;
- As expectativas devem estar alinhadas entre todos os participantes, incluindo a gerência. Expectativa, benefícios e limitações de um processo de automação de testes de *software* deve ser conhecidos por todos os envolvidos;
- Inicialmente o investimento na automação pode parecer alto, inclusive se comparado com o investimento necessário para realizar testes manuais. O retorno não é imediato, só é possível perceber os benefícios da automação a médio ou longo prazo, entretanto ainda assim a automação da execução dos testes apresenta bom ROI [17].

Automação de teste é o uso de *software* para controlar a execução do teste de outro *software*, a comparação dos resultados esperados com os resultados reais, a configuração das pré-condições de teste e outras funções de controle e relatório de teste. De forma geral, a automação de teste pode iniciar a partir de um processo manual de teste já estabelecido e formalizado.

Na prática a proposta de automatizar os testes visa diminuir o envolvimento humano com atividades repetitivas através da aplicação de estratégias e ferramentas [24]. Nunca a automação deve ser considerada com uma substituição aos testes manuais, as duas técnicas devem ser utilizadas em conjunto para aumentar a confiança na aplicação construída. Lembrando sempre que antes de iniciar um projeto de testes automatizados é imprescindível ter um processo de testes robusto e maduro [25].

Apesar do teste manual de *software* permitir encontrar vários erros em uma aplicação, é um trabalho maçante e que demanda um grande esforço em tempo. Também, pode não ser efetivo na procura de classes específicas de defeitos. A automação é o processo de escrita de um programa de computador para realizar o teste. Uma vez automatizado, um grande número de casos de teste podem ser validados rapidamente.

As vantagens da automação tornam-se mais evidentes para os casos de *softwares* que possuem longa vida no mercado, devido ao fato de que até mesmo pequenas correções no código da aplicação podem causar a quebra de funcionalidades que antes funcionavam. Entre essas vantagens estão a redução do tempo de execução e custo e aumento da cobertura e eficiência dos testes

A automação pode envolver testes de caixa-preta ou caixa branca. Para os dois casos, a cobertura de teste é determinada respectivamente pela experiência do desenvolvedor ou pela métrica de cobertura de código.

A automação de teste pode ser cara, e geralmente é usada em conjunto com técnicas manuais. Existem duas abordagens usadas na automação:

Baseado na *Interface* Gráfica

> No teste de *interface* gráfica, uma plataforma gera os eventos de entrada na *interface* do utilizador do sistema e observa as mudanças na saída. Várias ferramentas fornecem funcionalidades para gravar e reproduzir as ações do utilizador. Sendo aplicável a qualquer aplicação que use uma *interface* gráfica, esta técnica possui a vantagem de exigir menos codificação, mas implica dificuldades de manutenção do teste, como quando a *interface* gráfica é alterada.

Baseado no Código

> No teste baseado em código, a *interface* pública das classes, módulos ou bibliotecas são testadas com uma variedade de argumentos de entrada, observando-se a saída. Uma tendência atual no desenvolvimento de *software* é o uso de plataformas xUnit, que permitem realizar testes de unidade para determinar se as seções do código estão processando de forma esperada em diversas circunstâncias. O teste baseado em código é uma funcionalidade chave no desenvolvimento ágil de *software*.

6. TEST DESIGN

Nessa seção serão descritas os padrões e *designs* que foram utilizados durante todo o projeto e uma breve explanação sobre a decisão de utilização de cada um. Os padrões aqui citados são comumente utilizados em projetos de automação para melhorar a manutenção e extensibilidade dessas ferramentas [18] [24].

6.1 Record and Playback

Nessa técnica de automação de testes, o usuário grava com o auxílio de uma ferramenta de automação, as ações manuais realizadas na aplicação em um *script*, que pode ser executado em um momento posterior. Usando essa técnica será criado um *script* para cada caso de teste gerado para o sistema. Entre as técnicas apresentadas essa é a mais simples, porém mais difícil de manter e reaproveitar.

Algumas desvantagens no uso dessa técnica são: grande quantidade de *scripts* similares (duplicação de código), dificuldade na gestão de erros associados a interação da ferramenta de automação com a aplicação em teste, dificuldade na manutenção dos *scripts* e dificuldade para utilização dessa técnica em sistemas que não possuem GUI (Graphical User Interface).

6.2 Page Objects

Page Objects, ou padrão de páginas, é uma técnica muito utilizada no universo da automação de testes, cujo o intuito é melhorar a manutenção e evitar a duplicação de código. A ideia é criar um objeto para cada página *web* da aplicação que está sendo testada, dessa forma, o *framework* irá interagir com esses objetos sempre que necessitar realizar uma ação na *interface* gráfica [18].

Algumas das vantagens da utilização desse padrão são: caso haja alguma mudança na *interface* de usuário, os testes não são afetados. É necessário apenas atualizar as informações do objeto que representa aquela página, há uma separação clara entre o código do *framework* de testes e código específicos das páginas da aplicação e um único ponto passa a oferecer o serviço das páginas reduzindo a duplicação de código e facilitando a sua leitura.

Com a utilização do *Page Objects*, o ETA pode permitir que o usuário crie as classes de página da aplicação que ele deseja testar e ao mesmo tempo utilizar as funcionalidades providas pela ferramenta.

6.3 Data Driven

O *Data Driven*, ou Orientação a dados, é uma adaptação da técnica *Record and Playback*. Ele é utilizado para parametrizar os dados de um *script*, facilitando a substituição desses dados quando necessário. Seu intuito é resolver os problemas de parametrização de dados, duplicação de código e dificuldade de manutenção da técnica anterior [18].

Os *scripts* conterão parâmetros que carregarão de uma fonte de dados externa, os valores a serem utilizados. Dessa forma, é possível reexecutar os casos de testes dos *scripts* com diversos valores diferentes, sem ser necessário a criação de um nova sequência de passos para cada conjunto de dados que será utilizado.

Como desvantagem, continua sendo difícil gerir erros associados a interação da ferramenta de automação com a aplicação em teste e a adoção dessa técnica pode não ser vantajosa para casos em que a massa de dados seja reduzida. Os *scripts* conterão parâmetros que carregarão de uma fonte

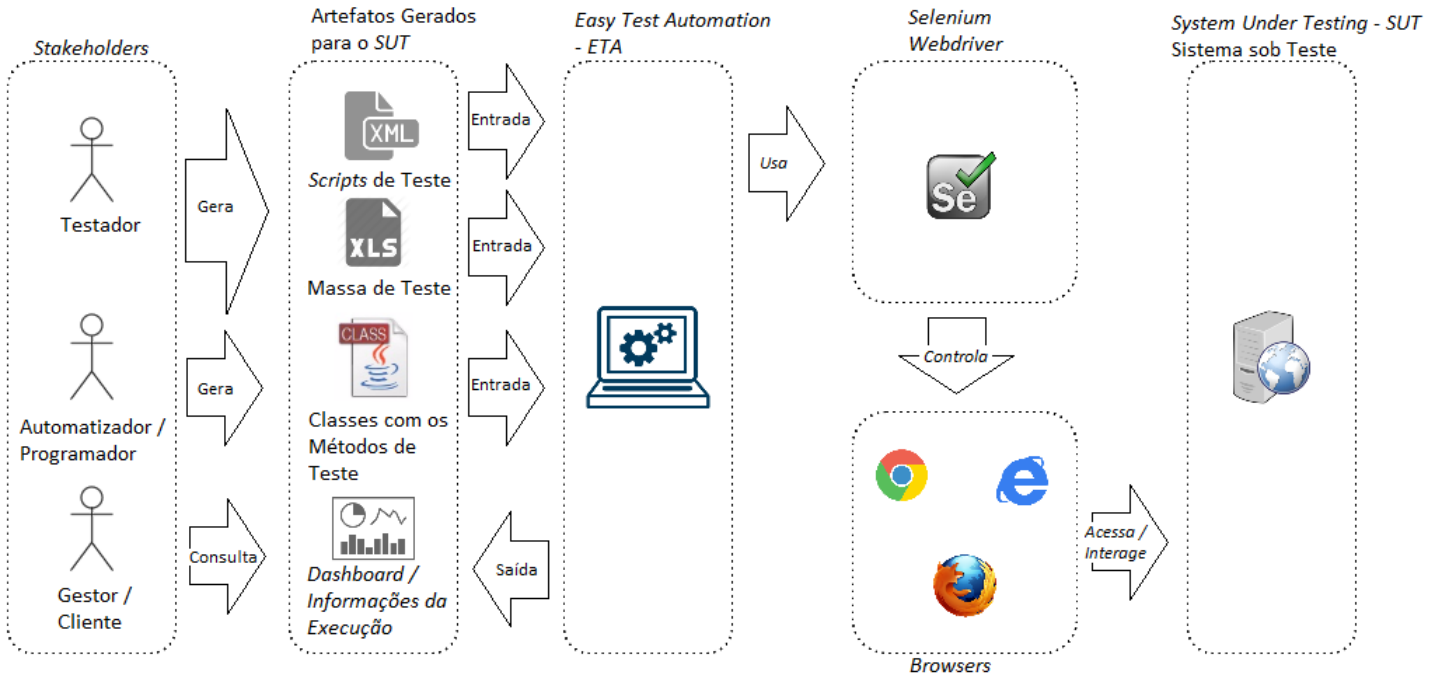

Figura 5: Interação entre os artefatos, *Stakeholders* e *Easy Test Automation*

de dados externa (planilhas, arquivos cvs, arquivos xml ou banco de dados), os valores a serem utilizados.

Para prover essa funcionalidade na ferramenta, foi utilizada anotação "@DataProvider" do TestNG [10] em conjunto com a biblioteca para a manipulação de planilhas de texto Apache POI [2]. Através da referida anotação o TestNG envia para os métodos os parâmetros capturados da planilha pelo Apache POI, permitindo assim a manipulação desses dados. O *Data-Driven* é considerada uma boa prática da automação de testes. Sua prática é aconselhada e valorizada em bons projetos e ferramentas de automação.

6.4 Key Word Driven

Nessa técnica, são identificados conjuntos de ações do usuário, que representam as suas interações com a aplicação. Essas ações são automatizadas apenas uma vez e podem ser reutilizadas. Utilizando essa técnica, há uma clara separação entre o que testar e como testar. Associada ao *Data Driven*, ela provê robustez e flexibilidade ao *framework* de testes. Para resolver a deficiência do *Data Driven* ela permite que sejam incluídas verificações de erros associados a interação do *framework* com a aplicação em teste.

Um desvantagem associada ao uso dessa técnica, é que ela requer um grande esforço inicial para ser implantada. Apesar que, uma vez em uso, ela é de fácil manutenção e provê maior facilidade no processo de criação de casos de teste.

7. ETA

A automação de testes é uma área dentro do processo de desenvolvimento de software que vem ganhado cada vez mais espaço, sendo utilizada por um maior número de empresas e pessoas. Apesar disto, ainda é uma área imatura comparada com outras áreas da Engenharia de Software [27].

Muitas vezes o sucesso da implantação da automação é alcançado através da tentativa e erro, apesar desta abordagem gerar um desgaste e desconfiança sobre os reais benefícios da implementação da automação.

Mitos e falta de conhecimento, geram percepções errôneas sobre os reais benefícios, limitações e gastos decorrentes da adoção de testes automatizados para um projeto.

Além disso, muitas vezes as expectativas não estão bem alinhadas entre a equipe de teste e a gerência do projeto, e a automação pode ser vista como a solução perfeita para resolver todos os problemas de qualidade do projeto, o que quase sempre irá gerar descontentamento, desconfiança e falta de credibilidade com o projeto de automação.

Visando garantir uma maior seriedade na abordagem e demonstrando de forma clara os benefícios e limitações dessa proposta, a ferramenta, descrita nesta seção, tem como principal foco facilitar e auxiliar na tarefa de criação de *scripts* e execução de teste automáticos pelos testadores.

Em geral, testadores não tem muita familiaridade com o desenvolvimento de aplicações/códigos. Em alguns casos o indivíduo faz a opção pela área de testes justamente para não ter contato diretamente com o código da aplicação, entretanto, não necessariamente o fato de testar um software, irá excluir a possibilidade de contato com o código da aplicação.

Esse fato, muitas vezes dificulta o aprendizado da automação e aumenta o desinteresse dos testadores pela área. Em um cenário que cada vez mais, as empresas buscam profissionais com conhecimento em automação e incorporam a prática ao processo de desenvolvimento das empresas brasileiras. Por isso, muitos deles não se interessam e desconhecem os benefícios que a automação da execução dos testes

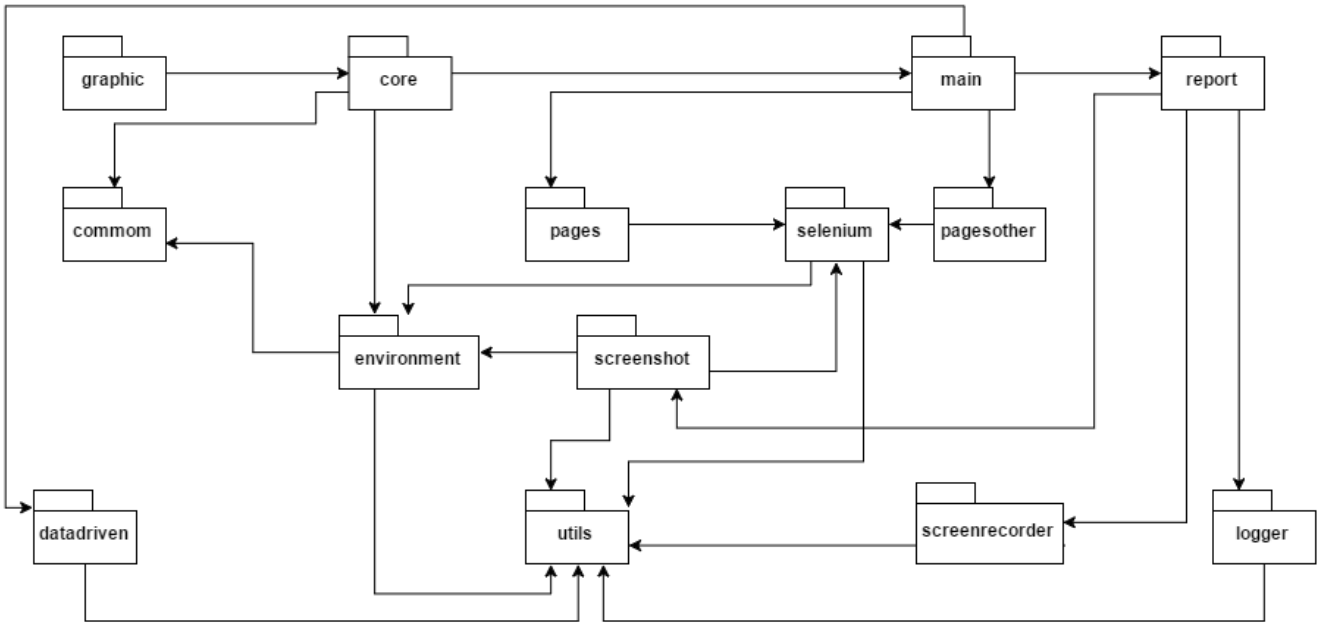

Figura 6: Diagrama de pacotes do ETA

pode trazer para eles, para o projeto que eles atuam e para a empresa.

Pensando nisso, surgiu a proposta de criar o ETA, uma ferramenta que proporciona os principais benefícios da automação de maneira simplificada, para que o testador possa escrever os *scripts*, sem se preocupar com a implementação propriamente dita e seus aspectos. É uma aplicação desenvolvida para utilização na execução automatizada de testes em aplicações exclusivamente *web*.

A ferramenta deve ser configurada e preparada por analistas de teste, automatizadores ou ainda testadores que possuam experiência com programação ou domínio sobre o ETA, o seu foco principal é na execução de testes funcionais de regressão, sistema e unidade, podendo utilizar tanto a técnica caixa branca, quanto caixa preta.

Foram utilizadas as técnicas de *Test Design Page Objects*, *Data Driven* e *Key Word Driven* na construção do ETA. Os *scripts* podem ser gerados por qualquer profissional que tenha conhecimento no funcionamento da ferramenta, desde testadores até gestores de projeto.

A Figura 5 apresenta a interação entre os diversos atores de um ambiente típico de uso do ETA.

Na versão 1.0, o ETA provê as seguintes funcionalidades:

- Criação de *scripts* simplificada através de arquivos XML;
- Captação de massa dados de arquivo externo (planilha do Excel);
- Geração de evidência em vídeo (*Screencast*);
- Geração de evidência em imagens (*Screenshot*);
- Geração de *logs* de mensagens e erros;
- Execução de testes em ambiente Windows 7 e Linux (Ubuntu);
- Execução de testes *Cross-Browser* (atualmente é possível realizar a execução nos *browsers* Internet Explorer 11, Mozila Firefox 45, Google Chrome 49 e Safari, incluindo as versões anteriores desses *browsers*);
- Execução de testes *Cross-Browser* em paralelo;
- Geração de relatórios detalhados com informações de execução, ambiente, evidências e gráficos;
- Possibilidade de execução via interface gráfica ou linha de comando.

O SUT (*System Under Testing*) utilizado durante os testes e validações do ETA foi o *Mercury Tours* [5]. Trata-se de uma aplicação criada pela HP para testes da sua solução de automação. No intuito de ajudar o entendimento, foi criada uma POC (Prova de conceito), para apresentar as principais funcionalidades disponíveis no ETA.

7.1 Arquitetura

O ETA foi elaborado de forma que a sua estrutura pudesse facilitar possíveis manutenções e extensões, para inclusão de novas funcionalidades. Assim foi realizada a divisão de pacotes por grupos de funcionalidades / responsabilidades relacionadas.

A Figura 6 apresenta o diagrama de pacotes da aplicação e seus relacionamentos. Devido a grande quantidade de classes e relacionamentos, optou-se por representar a estrutura do ETA conforme o diagrama. Nessa seção serão apresentadas individualmente as classes presentes nos pacotes da Figura 6 e seus respectivos métodos.

O ETA possui duas camadas principais: a primeira camada é a de negócio e fornece os serviços e funcionalidades ao testador/programador; a segunda contém as classes, que serão desenvolvidas para cada aplicação que será testada e que se utiliza da camada de negócio para realizar os testes no

SUT (*System Under Test*). Primeiramente será descrita a camada de negócio e suas classes, para posteriormente falar da parte que é responsabilidade do usuário do ETA.

O pacote `graphic`, apresentado na Figura 7 contém a *interface* (GUI - *Graphical User Interface*) com a qual o usuário irá interagir para configurar os parâmetros para execução. Além de possuir os componentes gráficos, o pacote possui a classe `TestNGRunner`, que é responsável por receber os parâmetros de execução do *TestNG* e fazer a chamada a classe principal para iniciar a execução dos testes. A classe `TestNGRunner` é quem carrega as informações do XML, caso o usuário tenha selecionado a opção de executar via linha de comando.

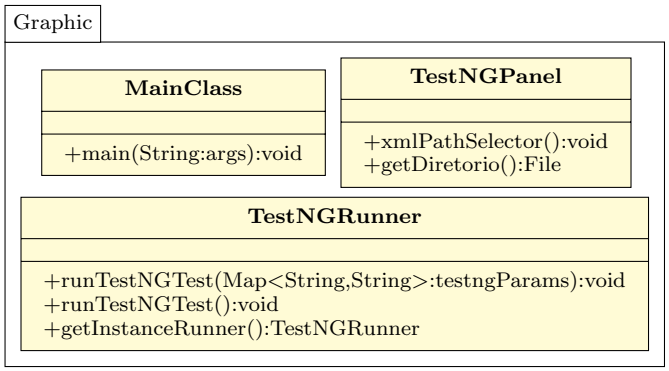

Figura 7: Diagrama do pacote `graphic`

No pacote `main` devem estar listadas as classes que irão implementar os métodos de teste, ver Seção A. No caso dessa POC a classe que contém os métodos de teste está representada na Figura 8, essa classe deve ser implementada pelo usuário do ETA (programador ou automatizador), ou seja, ela será diferente para cada SUT que utilizar o ETA como ferramenta de automação de testes.

Os pacotes `screenshot` e `screenrecorder` contém as classes responsáveis por realizar a captura das evidências em vídeo e imagem. A funcionalidade de *screenshot* foi pensada de forma que, a depender da plataforma a qual a aplicação esteja sendo executada, a ferramenta possa decidir qual o método mais adequado para capturar a evidência em *runtime*.

Para geração dos *screenshots* em ambiente *desktop*, é utilizado um recurso que o Selenium oferece através da classe `SeleniumUtils`, que disponibiliza uma *interface* para captura dos *screenshots* permitindo a quem está usando a classe determinar o formato de armazenamento da evidência, o local e as dimensões da imagem capturada.

Já as evidências em vídeo são capturadas com o auxílio da biblioteca *Monte Media Recorder* [28], ela permite que os vídeos sejam gravados no formato *Q*uickTime e AVI , além disso, disponibiliza gravação de vídeos com áudio e permite ajustar o tamanho da tela que será capturada.

Por padrão a biblioteca salva os vídeos na pasta *default* de vídeos do sistema operacional (no caso do Windows, na pasta "Meus Vídeos"). Entretanto, a classe responsável pela persistência do *Monte Media Recorder*, foi sobrescrita para permitir a seleção do local onde o vídeo será salvo e assim prover flexibilidade nesse aspecto.

O pacote `screenrecorder` contém as classes `SpecializedScreenRecorder` e `ScreenRecorderVideo` que são responsáveis pelo controle de *play/stop* dos vídeos e também pelas ações descritas anteriormente. Essa solução para gravação dos vídeos, só foi testada para ambientes *desktop*, para utilização em outras plataformas é necessário fazer um estudo da viabilidade. A Figura 9 apresentam as classes do pacote `screenrecorder`.

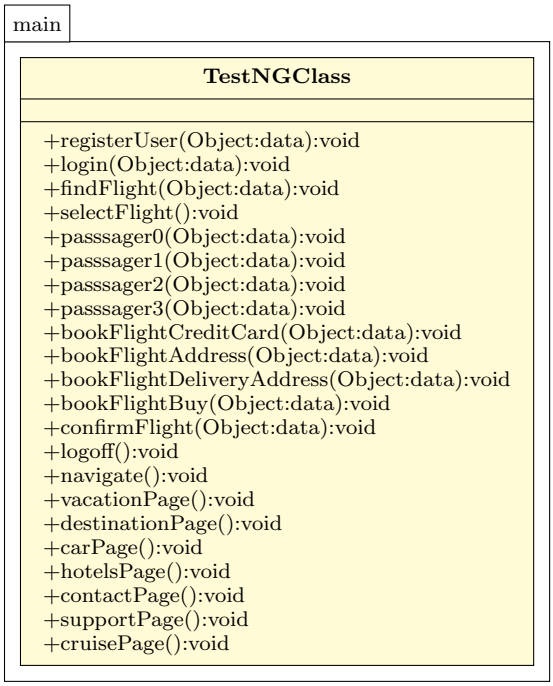

Figura 8: Diagrama do pacote `main`

Essa divisão dos pacotes específicos para evidências, é importante para manter a flexibilidade, uma vez que é possível que a depender da plataforma, a maneira de realizar os *screenshots* e *screencasts* sejam diferentes. Sendo assim, o ETA pode definir qual será a estratégia para capturar as evidências através dos métodos disponíveis nas classes `Screenshot` e `Screencast`, a depender do ambiente que ele esteja sendo executado. Essa flexibilidade foi possível através da utilização do padrão de projeto *Strategy*, além disso a classe `Screenshot` é um *observer* da classe `BrowserDrivers`.

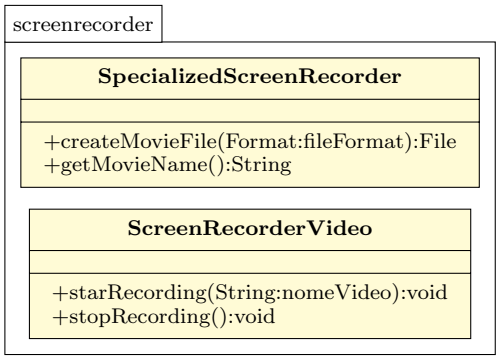

Figura 9: Diagrama do pacote `screenrecorder`

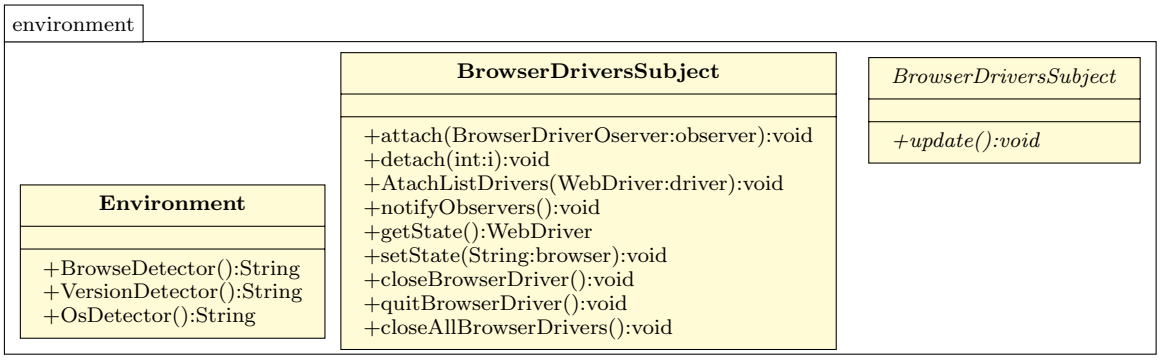

Figura 10: Diagrama do pacote `environment`

Em `environment` estão as classes responsáveis por fazer toda a manipulação dos *drivers* dos *browsers* e identificação do ambiente onde os testes estão sendo realizados (sistema Operacional, *browser* e versões), essas ações são realizadas pelas classes `BrowserDriversSubject` e `Environment` respectivamente.

A classe `BrowserDriversSubject` é implementada utilizando o padrão de projetos *Observer*, o intuito dessa implementação é permitir que *observers* sejam conectados a classe e informados quando o driver da execução atual for alterado. Isso se dá porque, para cada teste executado, o ETA inicia uma instância do *browser*, visando garantir o máximo de independência possível entre os testes. Os *observers* são derivados da classe abstrata `BrowserDriversObserver`. A Figura 10 apresenta a disposição dessas classes dentro do pacote.

O pacote `selenium` contém uma única classe a *SeleniumInteraction*, ela possui alguns métodos que encapsulam as interações do robô de automação com a aplicação que está sendo testada, além de ser um *observer* da classe *BrowserDrivers*. Dessa forma é possível que caso sejam utilizadas outras bibliotecas de automação, o impacto da mudança da biblioteca seria minimizado com a abstração implementada por essa classe, a Figura 11 apresenta o diagrama de classes do pacote.

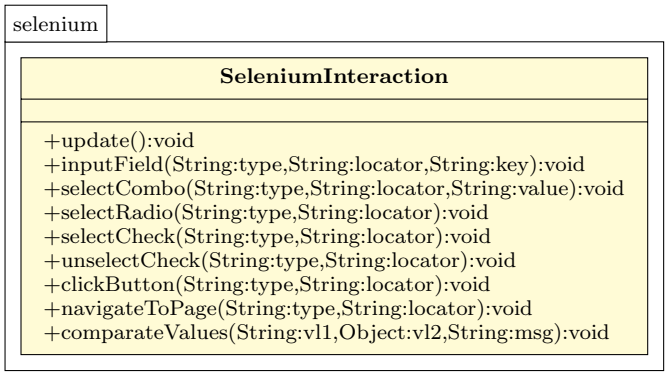

Figura 11: Diagrama do pacote `selenium`

A aplicação fornece para os *stakeholders* dois tipos diferentes de relatório, cada um com suas características e focados em apresentar informações de diferentes maneiras para os diferentes públicos, sendo assim pessoas que possuam conhecimentos técnicos ou não podem analisar e compreender os dados extraídos da execução dos testes.

O primeiro é o relatório de execução gerado pelo TestNG, este relatório é técnico e pode ser utilizado pelo desenvolvedor que fará a correção, caso o teste falhe e encontre algum erro no SUT. Ele apresenta o *stack trace* do erro, quantos testes foram executados, quantos falharam quantos passaram e se houve algum teste não executado. Ainda é possível verificar informações sobre a duração da execução.

O segundo relatório é gerencial, menos técnico, mais completo e rico em questão de informações do que o gerado pelo TestNG. Ele pode ser facilmente lido e entendido pelos clientes, pela gerência ou por outros membros da equipe do projeto. Para gerar este relatório, utilizamos a biblioteca *ExtentReports* [13].

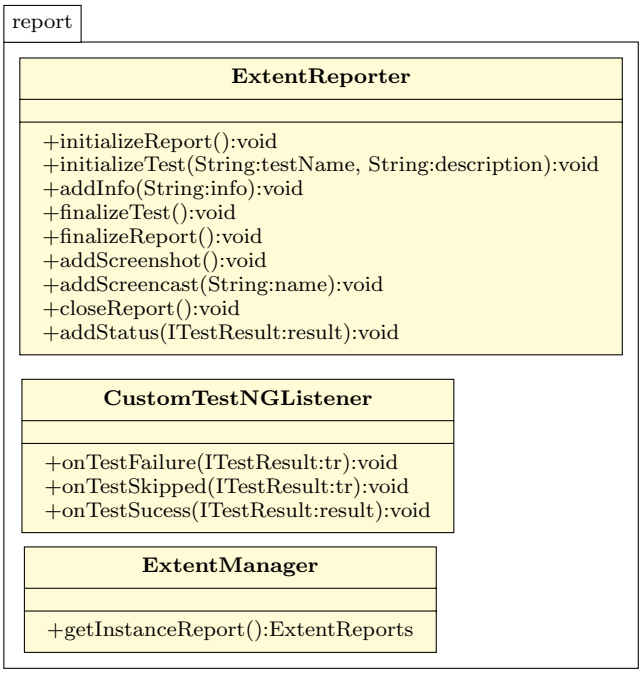

Figura 12: Diagrama do pacote `report`

Esta biblioteca provê um relatório HTML (Figura 19) con-

tendo informações sobre cada teste executado, duração, status, evidências e passos. Possui sessões para apresentação de detalhes do ambiente de execução, apresentação de conjunto de *Screshoots*, *ScreenCasts* e relatório de erros. Além de gerar gráficos em pizza contendo informações sobre total de testes executados e quantidade de testes por *status*.

O pacote `report` contém as definições e comandos necessários para geração dos relatórios que poderão ser gerados pela aplicação, lá são definidas as configurações para geração dos relatórios, caminho, conteúdo e demais informações. Esse pacote está definido na Figura 12.

O pacote `common` contém a classe `Property`, que é responsável por fazer a interação com o arquivo de propriedades e obter as configurações armazenadas nesse arquivo.

O pacote `datadriven` contém as funcionalidades responsáveis por realizar a captura e passar para os métodos de testes os dados que serão utilizados na execução. Estão disponíveis diversas formas de captura de dados, que podem ser selecionados de acordo com a necessidade da execução.

No pacote `logger` são definidas as configurações para os *logs* que a aplicação irá gerar do seu funcionamento e da execução dos testes, o pacote contém a classe `LoggerManager` que controla a criação e acesso aos *logs* gerados.Os *logs* gerados pela execução da aplicação, podem auxiliar no *debug* de falhas que não estão relacionadas ao teste, assim como erros associados a estrutura montada e as bibliotecas utilizadas. Os *logs* são gerados com o auxílio da biblioteca Apache log4j [1].

O pacote `core` é um dos responsáveis por iniciar a aplicação e realizar as chamadas iniciais para execução dos testes, ela interage com as classes do pacote `graphic` para iniciar a aplicação.

O pacote `utils` , contém as classes de auxílio a algumas funções úteis a toda aplicação. São elas:

- `ExcecelUtils`: Utilizada pelo aplicação para auxiliar na manipulação (leitura e gravação) das planilhas com os dados utilizados pelo pacote `DataDriven`;
- `FolderUtils`: Responsável por criar e controlar o acesso as pastas dos relatórios, *logs* e evidências;
- `SeleniumUtils`: Contém métodos de apoio para otimizar a codificação das classes de página (*screenshots*, verificação de elementos visíveis, habilitados, clicáveis dentre outros);
- `StringUtils`: Contém métodos para manipulação de *strings* (alteração de conteúdo, máscara e acentuação);
- `TestNGUtils`: Contém métodos que auxiliam na execução e parametrização do TestNG;

O usuário da ETA deve criar e manter as classes que são responsáveis por mapear os campos das telas do SUT, essas classes devem obedecer ao padrão *Page Objects*[11]. Para a POC feita nesse projeto foram criados dois pacotes que encapsulam todas as páginas disponíveis da aplicação testada, os pacotes `pagesother` e `pages`. A Figura 13 apresenta as classes do diagrama `pages`.

Nestas classes estão disponíveis os métodos responsáveis pela interação com SUT, que são utilizados pelo métodos da classe `main` a lista desses métodos e suas respectivas descrições estão disponíveis na Seção 12.A

[11]Ver Seção 6.2

O código XML abaixo, apresenta um exemplo de *script* do ETA. Esse *script* irá acessar o *Mercury Tours* e realizar a reserva de um vôo no *browser* Firefox de acordo com as informações carregadas da planilha de dados.

```
<?xml version="1.0" encoding="utf-8"?>
<suite name="suite" parallel="false">
 <listeners>
   <listener
    className="com.edu.ifba.logger.CustomListener"/>
 </listeners>
 <test name="Comprar passagem com sucesso">
  <parameter name="browser" value="firefox" />
   <classes>
     <class name="com.edu.ifba.main.test.TestNGClass">
       <methods>
         <include name="Login"/>
         <include name="findFlight"/>
         <include name="selectFlight"/>
         <include name="bookFlight"/>
         <include name="confirmFlight"/>
       </methods>
     </class>
   </classes>
 </test>
</suite>
```

8. FERRAMENTAS

Nesta seção serão descritas algumas soluções que foram utilizadas no projeto, para fornecer algumas das funcionalidades que o ETA disponibiliza.

Todas as soluções aqui apresentadas estão sob a licença *open-source*. Não foi incorporada nenhuma solução proprietária ao projeto.

8.1 Selenium

O Selenium é uma suíte/conjunto de ferramentas *open-source* para teste automatizados em aplicações *web*. O projeto está sob a licença Apache 2.0[12]. É possível escolher uma dentre diversas linguagens de programação, as quais o Selenium dá suporte, para criação dos *scripts*. Além disso, o Selenium suporta uma extensa quantidade de *browsers* e plataformas, esses foram alguns dos motivos que levaram a escolher o Selenium como ferramenta base para o desenvolvimento desse trabalho [18].

O Selenium é subdivido em três ferramentas distintas, Selenium IDE, Selenium WebDriver e o Selenium Grid. A biblioteca WebDriver(também conhecida como Selenium 2.0), utilizada como base para construção do ETA, surgiu da junção de outras duas ferramentas o Selenium 1.0 + WebDriver. Ela é a sucessora de outro membro da família Selenium o Remote Control. A Figura 14 apresenta as diferentes versões do Selenium [8].

Essa divisão das ferramentas disponibilizadas pela biblioteca Selenium, foi feita pensando no uso e características de cada um do tipos de ferramentas Selenium disponíveis. Na Figura 15 é apresentado um quadro descritivo com o relacionamento entre as diferentes versões.

O Selenium IDE (*Integrated Development Environment* - Ambiente Integrado de Desenvolvimento) é o ambiente gráfico de desenvolvimento de *scripts* de teste no Selenium. Ele

[12]http://www.apache.org/licenses/LICENSE-2.0

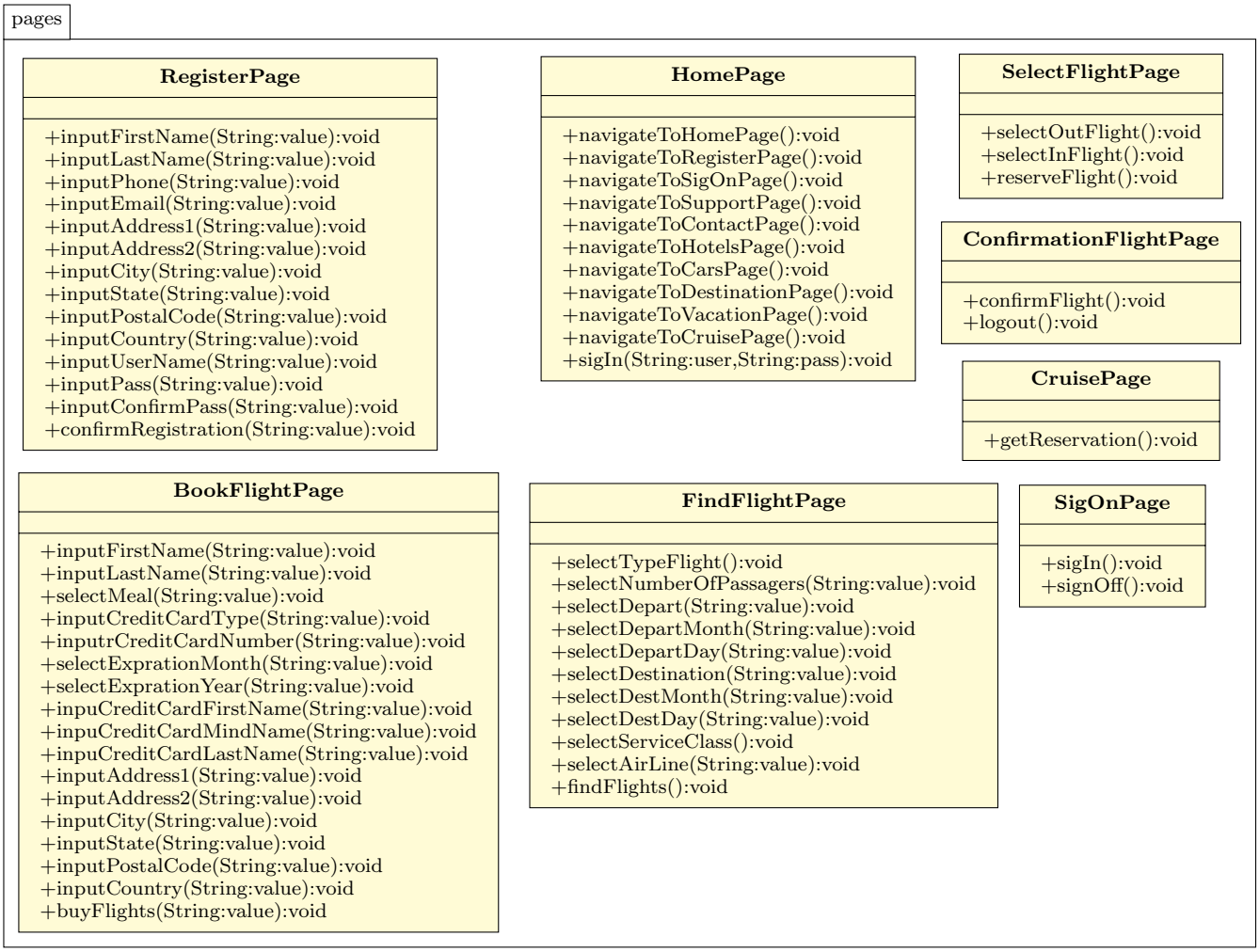

Figura 13: Diagrama do pacote pages

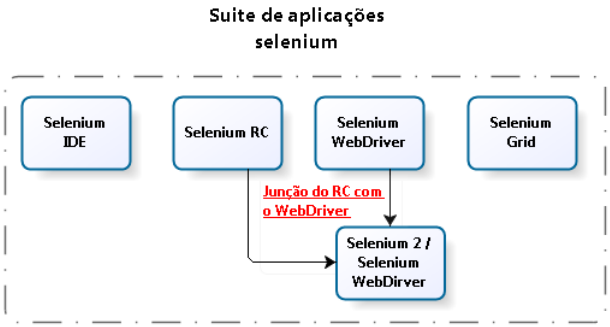

Figura 14: Suíte de aplicações Selenium

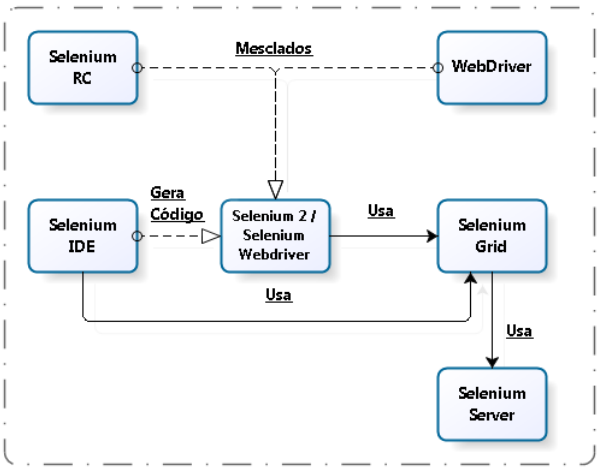

Figura 15: Interações entre as diferentes versões das ferramentas do pacote selenium

é implementado como uma extensão para *web browser*, disponível apenas para o Firefox. Ele utiliza o processo de *Record and PlayBack*, processo pelo qual a ferramenta grava todas as interações do usuário com o *browser* e as disponibiliza posteriormente para reprodução dos passos gravados.

Seu uso facilita esse processo, pois ele reconhece automa-

Tabela 2: Funcionalidades Selenium

Versão	Versão	Características
Selenium 1	Selenium RC	Ambos os nomes são utilizados para designar a mesma versão;
Selenium 2	Selenium WebDriver	Ambos os nomes são utilizados para designar a mesma versão;
Selenium RC	Selenium WebDriver	O WebDriver é o sucessor do RC; O WebDriver possui retro compatibilidade;
Selenium IDE	Selenium WebDriver	O IDE é uma ferramenta para captura de ações no Firefox, que possui possibilidade de exportação de código para o WebDriver; O WebDriver é um framework para automação de testes programaticamente;
Selenium Grid	Selenium WebDriver	O Grid é uma ferramenta para execução de testes do selenium paralelamente em diferente computadores (Execução distribuída); O WebDriver permite execução em apenas um computador;

ticamente os campos e seus identificadores, dessa forma não é necessário que o testador faça essa identificação manualmente.

Ao contrário de outras ferramentas disponíveis no mercado, o Selenium IDE não permite apenas que o usuário grave e execute os *scripts*, como uma IDE completa, ele apresenta um conjunto de funcionalidades que facilitam o desenvolvimento dos testes e permite que eles sejam exportados para diversas linguagens de programação, entre elas, Java, Phyton, Ruby e o próprio Selenês[13].

O Selenium WebDriver é um *framework* cuja a intenção é facilitar e prover mais flexibilidade ao usuário que deseja automatizar seus testes. Diferente do IDE o WebDriver possui uma biblioteca e um conjunto de drivers que permitem sua interação com diversos *browsers* e dispositivos móveis, essa interação pode ser escrita nas mais diversas linguagens de programação.

A integração entre o Selenium WebDriver e o Selenium IDE, funciona da seguinte maneira: com o IDE é possível gravar os *scripts* e gerar código em uma linguagem de programação, que em conjunto com as bibliotecas do WebDriver permitirão que sejam realizadas interações com os *browsers*.

A terceira e última parte do projeto Selenium é o Grid. Esse módulo permite que os testes sejam executados em vários *browsers* em diferentes máquinas (execução distribuída). Ao contrário dos módulos anteriores o Grid não gera *scripts* ele apenas é responsável por gerir o paralelismo das execuções em máquinas diferentes. Na Tabela 2, é apresentado um quadro comparativo com as principais características de cada versão do Selenium.

8.2 TestNG

O TestNG (*Test Next Generation*) é um *framework* de testes unitários baseada no JUnit e NUnit,[14] que além de fornecer as mesmas funções que o JUnit, inclui algumas melhorias e acrescenta novas funções. As funcionalidades disponíveis no TestNG são [16] [10]:

- Disponibiliza anotações em forma de Tags;
- Define diferentes formas de execução para ambientes *multi-thread*, é possível realizar a execução em uma *thread* única ou utilizando um *pool* de *threads*;

[13]O Selenese (ou Selenês) é a sintaxe dos comandos gerados pelo Selenium, onde um teste escrito em Selenese é basicamente composto por uma tabela contendo três colunas. Uma coluna é usada para operação e as duas restantes são usadas para os argumentos.

[14]NUnit é um *framework* de testes unitários voltado para a plataforma .NET.

Tabela 3: TesteNG x JUnit

TestNG	JUnit
Relatórios HTML	Relatórios HTML com o auxílio do ANT
Gera relatórios para apenas um caso de teste ou para uma suíte de testes completa	Só é possível gerar relatórios de suítes mas, não para casos de teste separados
Suporta mais anotações que o JUnit, o tornando assim mais flexível	Menos flexível do que o TestNG
Possui o conceito de "grupos" de execução	Não permite a criação de "grupos" de execução
Arquivo XML simples e de fácil entendimento	Arquivo XML complexo e de difícil entendimento
Configuração de múltiplas *T*hreads para execução paralela com o Selenium Grid	Não permite a configuração de *T*hreads para execução paralela

- Possibilita a execução de testes utilizando políticas;
- Suporta *Data Driven* com a anotação @DataProvider;
- Suporta execução com utilização de parâmetros;
- Disponibiliza execuções de teste em conjunto (suítes) ou separadamente (mais flexível);
- Suporta diversas ferramenta e *plugins* que auxiliam na execução dos testes;
- Provê paralelismo na execução dos testes;
- Facilita a configuração da execução através de arquivos XML.

O TestNG foi escolhido devido a sua melhora compatibilidade com *DataDriven*, execução de testes por suíte, execução de testes em paralelo e maior flexibilidade que essa ferramenta propicia para o desenvolvimento, frente ao JUnit. A Tabela 3 apresenta as principais diferenças entre o TestNG e o JUnit.

9. CARACTERÍSTICAS DE AMBIENTE DE EXECUÇÃO

9.1 Multi Plataforma

A aplicação foi projetada para ser executada tanto em ambientes Windows como em ambientes Linux, a única restrição que há para execução, é em relação ao *browsers* disponíveis em cada plataforma.

As funcionalidades disponíveis no ETA se comportam igualmente nos dois ambientes sem haver a necessidade de qualquer configuração adicional ou diferenciada. Em contrapartida, ele foi preparado para suportar automação em dispositivos móveis, a estrutura montada permite que seja identificado a plataforma a qual a ferramenta está sendo executada e a partir daí ajustar a sua execução para aquela plataforma em *runtime*.

9.2 Cross-Browser

O *Cross-Browser* é uma funcionalidade propiciada pela biblioteca Selenium WebDriver e seus *drivers*. Nativamente o WebDriver possui suporte ao navegador Firefox através do FirefoxDriver, além dele é possível utilizar *drivers* externos para uso com o Internet Explorer, Google Chrome e Safari.

O IEDriver, ChromeDriver e SafariDriver são os *drivers* adicionais que são utilizados na ferramenta. O Selenium possui ainda suporte a *Handless Browser*[15] através dos *drivers phantomJS* e do *HtmlUnitDriver* [8].

Para testes em dispositivos móveis o Selenium provê dois *drivers*, o selendroid que é utilizado para automação em dispositivos Android e o iOS-driver para os testes automatizados em dispositivos moveis do iOS.

Com o auxílio do *TesteNG* é possível realizar os testes nos diversos *browsers* executando uma única vez sequencialmente ou em paralelo. Ao invés de iniciar uma execução para cada *browser*.

Isso é possível informando no arquivo TestNG.xml quais os *browsers* que deverão ser executados ou passando essa informação por parâmetros de configuração.

10. VALIDAÇÃO

Esta seção apresenta o método utilizado para realizar os testes e validações do ETA [29].

O objetivo desta atividade foi avaliar a redução no tempo de execução, através do uso da automação em detrimento da realização de testes manuais, além de avaliar as características da ferramenta em conformidade com a NBR ISO/IEC 9126-1[23] e corretude dos testes realizados.

A hipótese levantada foi a seguinte: os participantes deveriam criar uma suíte de teste de regressão, com o menor esforço e maior corretude possível, visando garantir que a inclusão de novas funcionalidades, não impacte no que já está desenvolvido. Havendo impacto, a suíte deve ter sido construída de modo a identificar estes impactos sem a necessidade de interferência do testador.

A corretude da suíte, criada pelos participantes do experimento, foi avaliada com base em testes realizados em uma suíte padrão previamente criada com o mínimo de testes possíveis (oráculo), necessários para garantir a estabilidade do *Mercury tours*, após alguma mudança.

Através dessa comparação foi possível avaliar não apenas a cobertura da suíte de testes criada pelos participantes do experimento, mas também a corretude das tarefas realizadas.

Além disso, foram comparados os tempos de execução da suíte, manualmente e utilizando o ETA. A partir daí, foi feita uma análise em relação a estes tempos de execução para que fosse possível estabelecer a taxa de redução de tempo, entre a utilização das duas abordagens.

O grupo de testes utilizado foi composto por profissionais da área de desenvolvimento e teste de *software*, com os mais diversos níveis de conhecimento, variando de estagiários a analistas.

O grupo que possuía seis indivíduos foi distribuído em dois outros grupos, um com um indivíduo que realizou os testes manuais e outro com seis indivíduos que realizaram os testes automatizados utilizando o ETA (incluindo o individuo do Grupo 01).

Após a realização das atividades previstas, foi apresentado um questionário de avaliação de uso, críticas e sugestões de mudanças no ETA, para que os participantes pudessem expor a sua opinião e atribuir notas às características avaliadas da ferramenta.

10.1 Preparação

O *Mercury tours* foi selecionado como prova de conceito, após uma avaliação em três sites utilizados pela comunidade de testes, para aprendizagem dos conceitos e utilização do Selenium.

Foram avaliados os aspectos de estabilidade da aplicação, facilidade e flexibilidade de uso e documentação disponível.

O *Tools QA* não possui documentação disponível e a utilização do site não é intuitiva, apesar de ser uma página de mercado eletrônico. Além disto, apresenta baixa flexibilidade de uso e é bem estável [6].

O Selenium *Begginners* é uma página criada para realização de exercícios do *Selenium Cookbook*. A página não contém um fluxo de ações, apenas validações e interações com componentes isolados, sem um fluxo de utilização bem definido. A página não é estável: durante a navegação é possível identificar diversos erros de carregamento e interação na página [7]

O *Mercury Tours* é mantido pela *Hewlett-Packard*(HP) para demonstrações de uso da sua solução para automação de testes. Apesar de não ter documentação disponível, a aplicação é intuitiva, simples de navegar e possui flexibilidade na realização de comandos e registro de usuários [5], por esse motivos ela foi escolhida para ser utilizada na POC do ETA.

Para auxiliar na realização dos testes pelos participantes do experimento, foram criados dois casos de uso do *Mercury Tours*, disponíveis na Seção 12.B.

Para realização dos testes, inicialmente foi criado um oráculo, a suíte padrão de testes necessários para serem realizados no *Mercury Tours*, que é composta pelos seguintes casos de teste:

- Navegar no sistema - navegação simples entre as funcionalidades disponíveis no *Mercury Tours*;
- Registrar usuário - realizar o registro de um novo usuário para utilização no *Mercury Tours*;
- Realizar uma reserva para um passageiro - realizar uma reserva de um voo para um passageiro no *Mercury Tours*

[15]Handless Browser é a execução dos testes *web*, onde a interação é realizada através do *Webkit* sem que haja necessidade de uma *interface* gráfica para interação com o *browser*.

- Realizar uma reserva para mais de um passageiro - realizar uma reserva de um voo para mais de um passageiro no *Mercury Tours*
- Realizar uma reserva para um voo, com pagamento fora dos Estados Unidos - realizar uma reserva de um voo, com o pagamento selecionado para um país diferente dos Estados Unidos.

Esses casos de teste foram escolhidos, por contemplar todos os fluxos disponíveis no caso de uso e também para garantir que os testes sejam realizados em todas as funcionalidades que já foram desenvolvidas e disponibilizadas no *Mercury Tours*.

Os grupos de participantes receberam tarefas que foram executas com supervisão do autor desse trabalho. Ao primeiro grupo foram dadas as seguintes tarefas: (i) leitura de caso de uso do *Mercury tours*, (ii) entendimento e navegação no *Mercury tours*, (iii) realização de testes exploratórios [12].

Ao grupo responsável por realizar a interação com o ETA, foram solicitadas as seguintes ações: (i) leitura de caso de uso do *Mercury tours* e lista de métodos disponíveis no ETA, (ii) entendimento e navegação no *Mercury tours*, (iii) questionamentos básicos sobre o funcionamento do ETA, (iv) criação da suíte e execução dos testes utilizando o ETA e (v) preenchimento do questionário de avaliação da ferramenta.

10.2 Questionário

O questionário criado possui questões referentes a entendimento, interação e utilização do ETA, além de perguntas relacionadas à norma NBR ISO/IEC 9126-1 de qualidade de *software*, que descreve um modelo de qualidade de produto de *software*, composto de duas partes: qualidade interna e externa; e qualidade em uso [23].

O intuito desse formulário foi coletar o *feedback* dos participantes do segundo grupo, e permitir avaliar o nível de utilidade, entendimento e qualidade da abordagem e da ferramenta desenvolvida.

Foram consideradas cinco características da norma de qualidade de software NBR ISO/IEC 9126, para elaboração do questionário, são elas [23]:

- Funcionalidade: capacidade de um *software* prover funcionalidades que satisfaçam o usuário em suas necessidades declaradas e implícitas, dentro de um determinado contexto de uso;
- Confiabilidade: capacidade do produto se manter seu nível de desempenho nas condições estabelecidas;
- Usabilidade: capacidade do produto de *software* ser compreendido, seu funcionamento aprendido, ser operado e ser atraente ao usuário;
- Eficiência: capacidade que mede se o tempo de execução e os recursos envolvidos são compatíveis com o nível de desempenho do software.
- Manutenibilidade: capacidade (ou facilidade) do produto de software ser modificado, incluindo tanto as melhorias ou extensões de funcionalidade quanto as correções de defeitos, falhas ou erros.

Essas características foram selecionadas para exemplificar o nível de qualidade de *software* associado ao ETA. A avaliação não foi completa e não seguiu todas as características definidas na norma, o intuito de basear o questionário nessas características foi ter uma visão básica e geral da ferramenta do ponto de vista da norma de qualidade de software NBR ISO/IEC 9126.

O formulário utilizado está disponível no endereço: https://goo.gl/YaEfmi

10.3 Resultados

Os dados apresentados nessa seção estão relacionados as tarefas que foram realizadas pelos grupos participantes desse experimento.

Grupo 01: Testes Manuais

Esse grupo foi composto por um único indivíduo com menos de um ano de experiência na realização de testes manuais e sem conhecimento sobre a área de automação de testes.

As duas primeiras atividades desse grupo foram: (i) leitura de caso de uso do *Mercury tours*, (ii) entendimento e navegação no *Mercury tours*. Essas atividades duraram ao todo cerca de 30 minutos. O participante teve a oportunidade de tirar dúvidas sobre o funcionamento da aplicação sob teste e dos fluxos presentes no caso de uso.

A terceira atividade consistiu na realização de testes exploratórios, após as primeiras atividades. O participante selecionou uma lista de situações que gostaria de testar e executou os testes com base nessa lista. Foram avaliados três cenários de execução:

1. Registrar um usuário com sucesso;
2. Realizar navegação até a tela de reserva de hotéis;
3. Reservar um voo com sucesso.

O tempo gasto para a execução do primeiro teste foi de três minutos, para o segundo um minuto e para o terceiro teste seis minutos, totalizando dez minutos para a execução da suíte de teste.

A taxa de corretude desse experimento foi de 60%, considerando que foram executados três casos de teste, que estavam presentes no oráculo.

Grupo 02: Testes Automatizados

O perfil do grupo que realizou os testes, era composto por profissionais com diferentes níveis de conhecimento e cargos, como pode ser visto na Figura 16.

Para realização dos testes com o ETA foi dado preferência a profissionais com perfil de analista júnior ou abaixo, apesar de terem também sido realizados testes com profissionais com uma senioridade maior.

Na Figura 17 são apresentados os dados referentes aos conhecimentos em teste de *software* e automação de testes dos participantes do Grupo 02. A maioria dos candidatos tem menos de dois anos de experiência com testes e metade dos participantes já teve algum contato ou experiência com automação de testes de *software*.

Após uma análise das suítes de teste criadas pelos participantes, chegou-se a conclusão que a taxa de corretude dos testes criados pelos analistas plenos chegou a 90%, enquanto que para os cargos de menor senioridade a taxa de corretude ficou em torno de 65%.

A lista a seguir, apresenta o tempo médio de execução dos casos de teste, pelo ETA:

Perguntas sobre o ETA						
Perguntas/Escala	**SIM**			**NÃO**		
Teve dificuldade para entender o funcionamento do ETA?				6		
Teve dificuldade para Criar os scripts?				6		
Teve dificuldade para Executar os scripts?				6		
Perguntas/Escala	**1**	**2**	**3**	**4**	**5**	**6**
Qual seu grau de satisfação com a utilização da ferramenta? (1-pouco satisfeito; 6-Satisfeito)					5	1
Em uma escala de 1 a 6, qual você acha que é o grau de utilidade do ETA para testes automatizados? (1-Pouco útil; 6-Muito útil)					3	3
Qual a probabilidade de você utilizar uma abordagem para automação de testes, como a apresentada pelo ETA? (1-Pouco provável; 6-Muito provável)					2	4
Em uma escala de 1 a 6, você acha que as funcionalidades disponíveis no ETA, atendem as necessidades da automação de testes? (1-Pouco apropriado; 6-Muito Apropriado)				1	3	2
Perguntas referentes a NBR ISO/IEC 9126						
Em uma escala de 1 a 6, você acha que o ETA faz o que foi proposto de forma correta? (1-Pouco correto; 6-Muito correto)					4	2
Em uma escala de 1 a 6, com que frequência você acha que o ETA apresenta falhas? (1-Poucas falhas; 6-Muitas falhas)	4	2				
Em uma escala de 1 a 6, como você avalia a facilidade de uso do ETA? (1-Muito difícil; 6-Muito fácil)					3	3
Em uma escala de 1 a 6, como você avalia o tempo de execução dos testes pelo ETA? (1-Muito lento; 6-Muito rápido)					3	3
Em uma escala de 1 a 6, como você avalia o esforço necessário para realizar alterações nos scripts? (1-Pouco esforço; 6-Muito esforço)	5				1	

Tabela 4: Respostas dos participantes do experimento no formulário

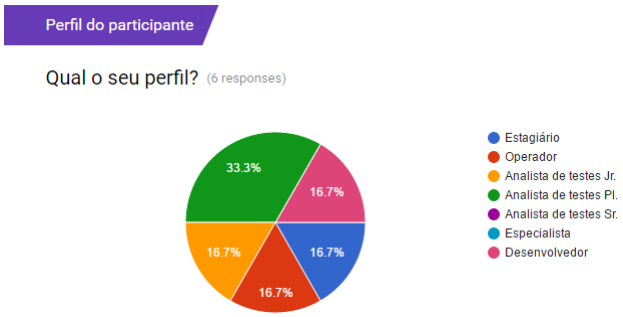

Figura 16: Gráfico de participantes por cargo

- Para os casos de teste de registro de usuário a média do tempo de execução do ETA foi de 20 segundos;
- Para realizar a navegação por todo o sistema o tempo médio de execução foi de 1 minuto e 30 segundos, (considerando a navegação por mais de uma ou por todas as telas disponíveis no sistema);
- Para as situações em que a reserva de passagens era realizada para apenas um passageiro o tempo médio de execução foi de 1 minuto. Quando havia mais de um passageiro o tempo médio corresponde ao acréscimo de dois segundos para cada passageiro, a esse tempo;
- Para o caso que a compra de passagem era realizada para fora dos Estados Unidos, o tempo médio de execução foi de 1 minuto e 05 segundos.

Figura 17: Gráfico de participantes por conhecimento em testes e automação de testes

Foi observado também diferença nos tempos de execução de um mesmo caso de teste para diferentes *browsers*, em geral o Internet Explorer foi o *browser* cuja a ferramenta demorou mais tempo para realizar as interações, chegando a levar mais que o triplo do tempo na execução de um mesmo caso de teste, em comparação ao que foi executado no Firefox e no Google Chrome. Isso se dá pela diferença na forma como o Selenium interage com os diversos *webkits* dos

browsers disponíveis no mercado [11].

A Figura 18 apresenta o relatório do TestNG para uma execução, que ilustra a diferença de tempo na execução do mesmo teste para três *browsers* diferentes. Nela é possível verificar que o *browser* cuja a execução ocorre mais rápido é o Google Chrome, seguido pelo Firefox e pelo Internet Explorer como o *browser*, cuja a execução durou mais tempo.

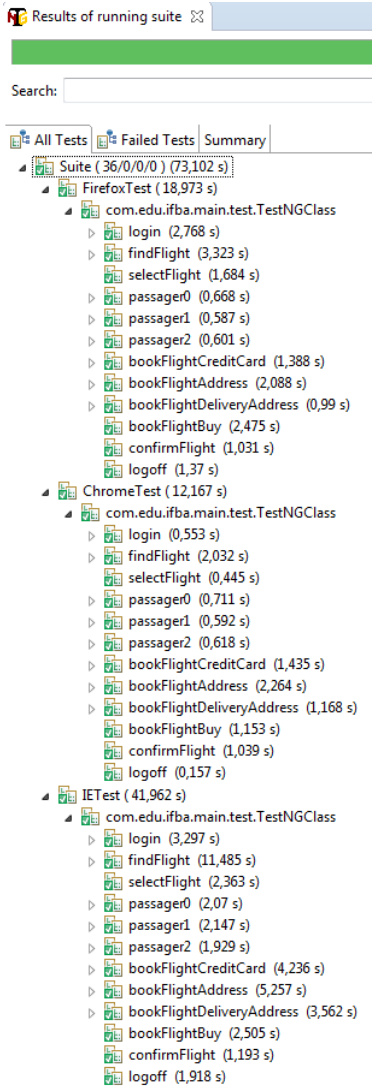

Figura 18: Resultado da execução do TestNG

A Figura 19 e a Figura 20, apresentam um exemplo de um relatório gerado pelo Extent Reports, para a execução de três testes, nele é possível ver quais os casos de teste foram executados, seus status, tempo de execução e evidências associadas (*Screenshots* e *Screencasts*).

No geral, os participantes demonstraram estar satisfeitos com a abordagem para automação de testes e funcionalidades disponíveis no ETA, não relataram ter problemas no entendimento da utilização de ETA, criação de scripts e execução dos testes.

A Tabela 4 apresenta um resumo das respostas dadas pelos participantes no questionário sobre a utilização do ETA e ítens referentes a norma de qualidade de software NBR ISO/IEC 9126-1 [23]. Nela é possível observar a quantidade de respostas obtidas para cada item do formulário. Vale salientar que nenhum dos participantes reportou dificuldade para criar *scripts* e executar a aplicação.

Com as validações realizadas foi possível verificar que em alguns caso a taxa de redução do tempo da execução chegou a 90% em relação a execução do teste manuais. A taxa de corretude se mostrou maior, para o grupo com maior maturidade e conhecimento sobre testes. O tempo de execução está diretamente ligado equipamento utilizado para os testes, foram usados dois *notebooks* com diferentes configurações.

Um HP com 4GB de memória Ram, processador i3 de segunda geração e sistemas operacionais Windows 7 e Ubuntu 14. Os testes realizados nesse equipamento, apresentaram maior tempo de execução, inclusive causando erros no ETA e na execução dos *scripts* devido a problemas de lentidão na interação com os *browsers*.

Por esse motivo, os testes passaram a ser realizados em um segundo equipamento. Um Lenovo com 8GB de memória Ram, processador i5 vPro e Windows 7. O tempo de execução nesse *notebook*, foi inferior em comparação com o equipamento anterior. Todos os testes apresentados nessa seção foram realizados nesse equipamento, inclusive repetindo os testes que haviam sido realizados no equipamento anterior.

Como pode ser visto na Figura 21, são apresentadas perguntas e respostas sobre a experiência dos participantes com automação de testes (caso possuam) e sua opinião a respeito da ferramenta apresentada. Algumas das sugestões de melhorias apresentadas pelos participantes foram a existência de uma interface gráfica para utilização do ETA, e geração de outros tipos de relatórios.

Quanto a experiência com automação de testes, foi possível verificar que alguns candidatos possuíam experiências acadêmicas ou com projetos piloto, mas nenhum teve contato direto com uma ferramenta com uma abordagem para automação de testes como o ETA.

11. CONCLUSÃO

Este trabalho apresentou o ETA - *Easy Test Automation*, ferramenta desenvolvida para auxiliar a equipe de testes no processo de automação de testes funcionais para aplicações *Web*.

O ETA permite que os *scripts* sejam criados de maneira simples, separando os comandos de teste dos dados utilizados para realizar os testes, provê relatórios técnicos e gerenciais, detalhados com evidências em *Screenshots*, *Screencasts* e *logs* de execução.

A ferramenta auxilia e aumenta a produtividade da execução de testes automatizados em aplicações *Web*, entretanto, de nada adianta ter centenas de testes, de processos de negócio e ferramentas de suporte ao teste e não conseguir identificar o que precisa ser testado a cada mudança. Por isso, é imprescindível que se adotem técnicas, processos e boas práticas para realizar os testes, pois, a automação de testes por si só, não pode garantir a qualidade do software que está sendo desenvolvido.

O *Easy Test Automation* está disponível para acesso no github no endereço https://github.com/amaralrfl/tcc.

12. TRABALHOS FUTUROS

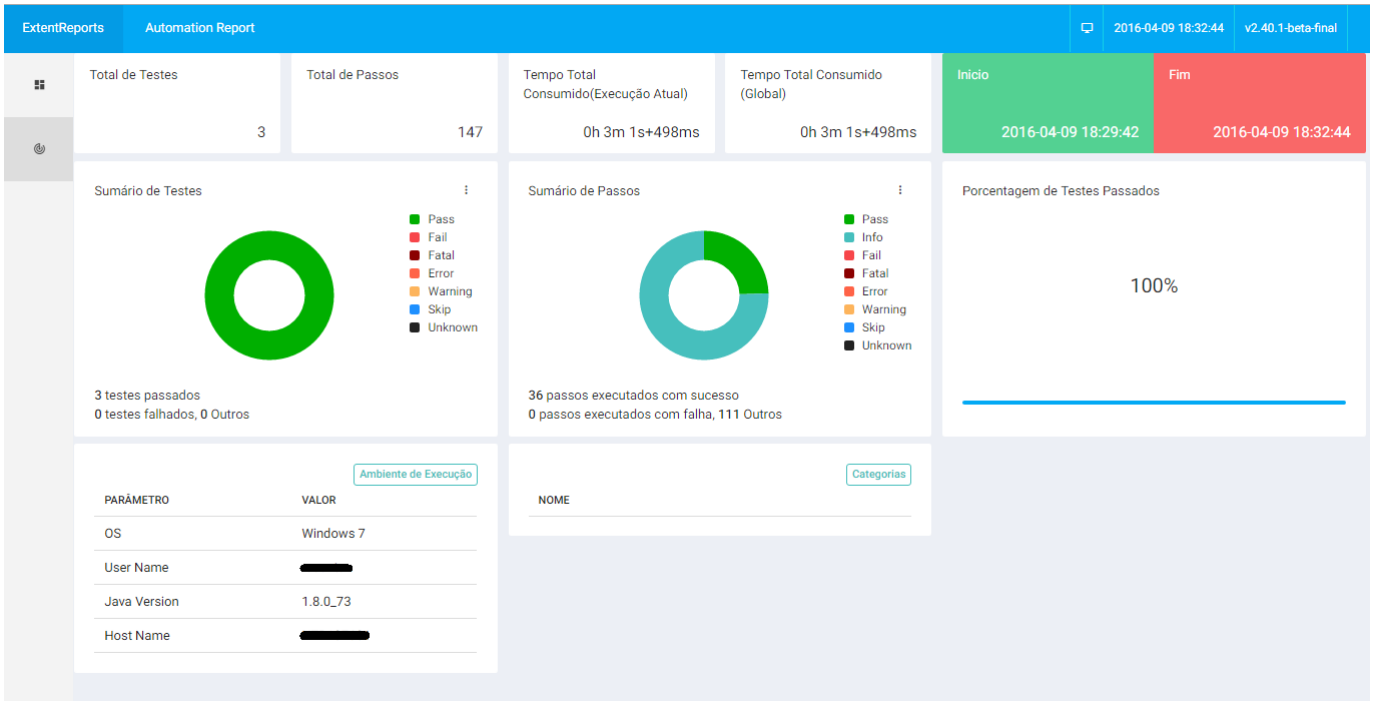

Figura 19: Relatório de execução no *Dashboard* do Extent Reports

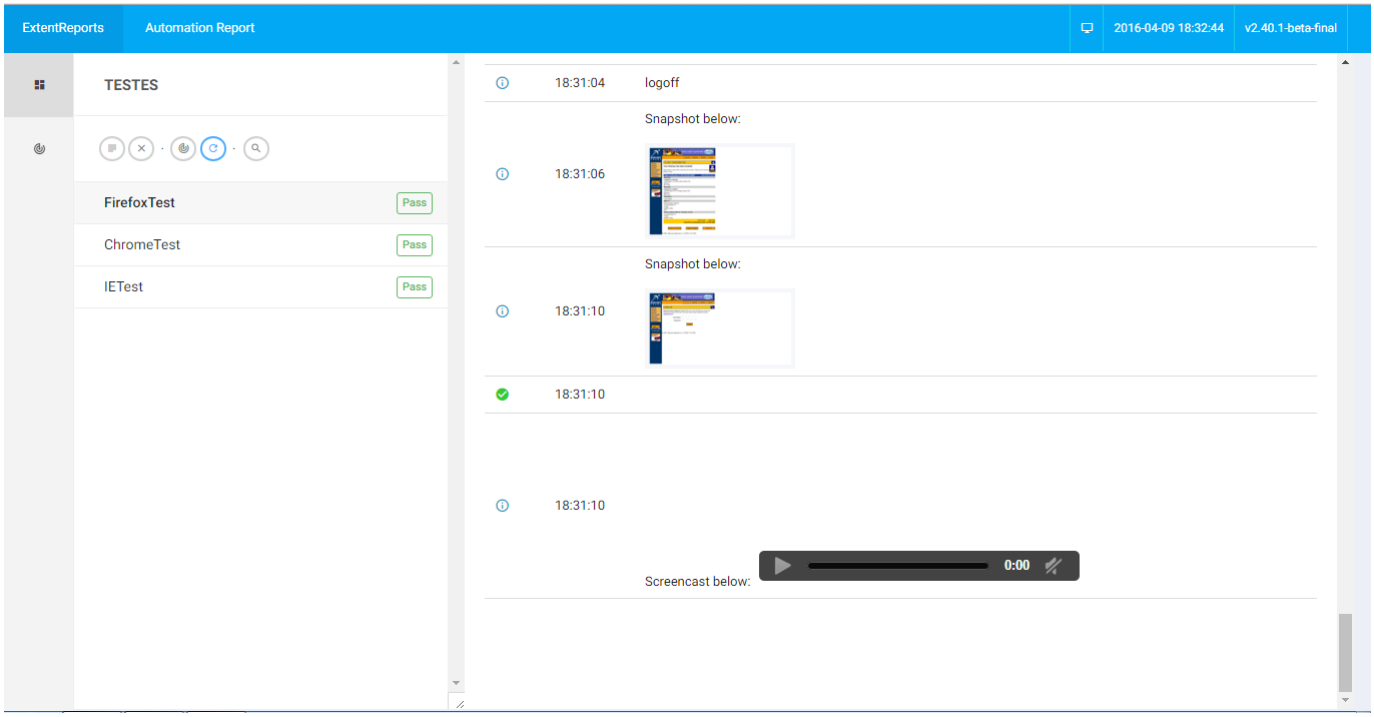

Figura 20: Relatório de execução dos casos de teste e evidências no Extent Reports

Aqui serão apresentados algumas funcionalidades desejáveis e que podem ser desenvolvidas para a enriquecer a aplicação. Nessa primeira etapa elas não foram contempladas no escopo do projeto e também não foram abordadas por restrições de tempo e/ou esforço mas são importantes temas para trabalhos futuros.

Integração com o TestLink [16]

[16] O TestLink é uma ferramenta para gestão dos testes. Ele tem como principal função organizar a elaboração, planejamento e execução dos casos de teste. Permitindo a associação de requisitos, casos de teste, ambientes de execução e ocorrências registradas em um *BugTracker*.

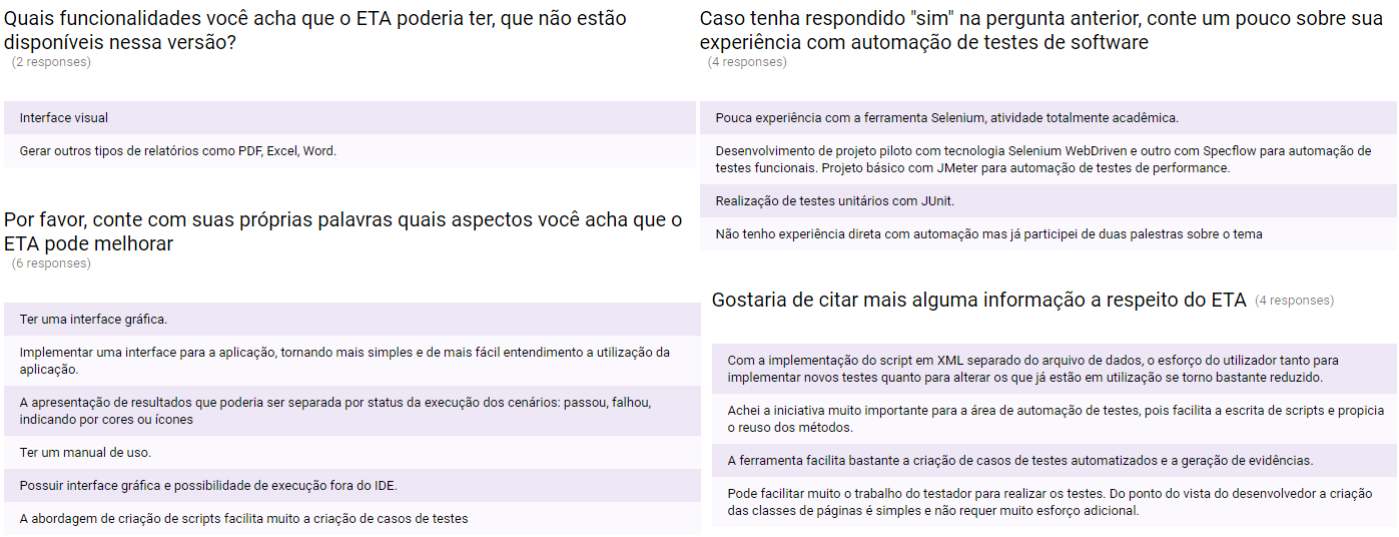

Figura 21: Perguntas realizadas no questionário

O Testlink como ferramenta utilizada para gestão e armazenamento dos testes, permite ao analista ou gestor ter uma visão real e em tempo real do andamento dos testes. Com todos os casos de teste cadastrados na ferramenta para execução e podendo realizar rastreamento de requisitos x casos de teste x defeitos encontrados.

Ele se apresenta como uma importante ferramenta para a equipe de testes gerando gráficos, provendo rastreabilidade e informações detalhadas sobre todos os testes realizados e informações cadastradas na ferramenta.

A ideia dessa integração é fornecer todas essas vantagens que o Testlink apresenta, também para os testes automatizados. O Selenium permite essa integração com a ferramenta, executar os testes a partir do Testlink e registrar os status da execução dentro do próprio Testlink. O trabalho futuro é estudar essa API para poder agregar esse novo serviço.

Envio de e-mails em caso de erro e fim de execução

A intenção nesse tópico é fornecer *feedback* rápido sobre a execução e os erros encontrados. O TestNG permite o envio de email com base em algumas métricas, a intenção é por exemplo, indicar para a ferramenta que quando ocorrer um erro "impeditivo" o testador ou os *Stakeholders* sejam imediatamente informados sobre a falha que ocorreu.

Integração como Jenkis para *build* automático[17]

O Jenkins é uma importante ferramenta utilizada para *build* automático e integração continua. A intenção dessa integração é permitir que a cada *deploy* realizado ou em um tempo pré-determinado os testes sejam executados automaticamente (sem a necessidade que alguém dê o "*start*" na execução).

Possibilidade de execução via linha de comando

Permitir que a aplicação seja iniciada via linha de comando e/ou arquivo .bat. Essa melhoria está fortemente relacionada com a integração com o Jenkins, uma vez que para realizar a execução o Jenkins chamará a aplicação via linha de comando.

Integração com o mantis bug tracker[18]

Alinhado com a integração com o Testlink, é desejável a integração como Mantis para *report* automático e registro das ocorrências que forem encontradas. Um detalhe importante que precisa ser registrado nessa melhoria é como será feita a análise do erro, se é um erro válido ou um falso positivo.

É necessário pensar bem nesse aspecto para não gerar desperdício de tempo e esforço analisando ocorrências que são falsos positivos. Provavelmente será necessário um testador analisar os erros encontrados antes de permitir que a ferramenta faça o registro das ocorrências no Mantis.

Execução em dispositivos móveis

A crescente utilização de dispositivos móveis exigiu dos times de desenvolvimento e testes uma readequação na forma de produzir os *softwares*. Essa nova abordagem trouxe novos desafios e necessidades ao processo, não seria diferente no que diz respeito a automação de testes.

A execução de testes automatizados em dispositivos móveis, é um requisito bastante desejado para essa aplicação. Por ser uma disciplina relativamente nova no âmbito do desenvolvimento de software, a criação de aplicações móveis e teste dessas ferramentas ainda não está madura. Levando em consideração a quantidade de dispositivos e sistemas operacionais móveis disponíveis no mercado, a tarefa de testar o sistema pode ser tornar dispendiosa e cara.

[17] O Jenkins é um servidor de integração continua que é utilizado para a realização de *deploy* automático

[18] O Mantis *BugTracker* é uma ferramenta *web*, cuja sua principal função é permitir o gerenciamento de ocorrência e defeitos dentro de um projeto de software.

Pensando nisso, a automação dos testes em dispositivos móveis se apresenta como uma forte pratica para garantia de qualidade dos sistemas mobile. Algumas iniciativas como o Selendroid, Iosdriver, Calabash e o Appium já fornecem interfaces compatíveis com o Selenium para execução de testes em aplicações nativas, híbridas ou *mobile web apps(HTML)* em dispositivos móveis Android, IoS e FirefoxOS.

A ideia dessa extensão é permitir que os *scripts* gerados, sejam executados também em dispositivos móveis, sem que seja necessária nenhuma alteração no código do *Easy Test Automation*.

Data Driven utilizando outras fontes de dados

Além da captura de informações do arquivo Excel, é possível carregar dados de outras fontes, para utilização. Por exemplo, uma outra possível fonte de dados que o TestNG suporta é de arquivos .db. A ideia é fornecer a possibilidade que o ETA vá buscar na base de dados a massa de dados necessária para a execução do teste. Ainda é possível utilizar arquivos CSV ou XML para *Data Driven*.

Execução paralela utilizando o Selenium Grid

O TestNG permite a execução paralela dos testes em uma mesma máquina física. Foram realizados testes utilizando esse tipo de abordagem, e aplicação obteve bons tempos de resposta e corretude nos testes executados.

O objetivo de utilizar o Selenium Grid para poder paralelizar a execução em outras máquinas, sejam elas físicas ou virtuais. É diminuir o tempo de execução de uma suite de testes muito grande ou executar determinada suíte em diferente plataformas (Windows, Linux) e *browsers* em paralelo.

Geração de relatórios em PDF

Outra opção de relatório que pode se adicionada a ferramenta é a criação de arquivos em PDF, contendo informações detalhadas sobre a execução, erros e *logs* gerados pela aplicação.

Execução em ambiente MacOS desktop

A aplicação foi testada apenas nos ambientes Linux e Windows, por isso, ainda não é possível determinar qual será o seu comportamento do ETA em ambientes MacOS. As funcionalidades que podem sofrer impacto com a mudança de ambiente (geração de evidências, relatórios, *logs*), estão preparadas para serem modificadas e caso necessário adicionar novos comandos de acordo com o sistema operacional que foi identificado pela ferramenta.

Testes de aceitação

A ideia é integrar a o ETA com um *framework* para BDD (Cucumber ou JBehave), dessa forma será possível criar teste automatizados de aceitação. Aproveitando todas as vantagens do BDD, além de prover a possibilidade de geração de *scripts* em linguagem natural.

13. REFERÊNCIAS

[1] Apache log4j. `http://logging.apache.org/log4j/2.x/`. Acessado: 22-01-2016.

[2] Apache poi project. `https://poi.apache.org/`. Acessado: 25-01-2016.

[3] Ibm - rational functional tester. `http://www.ibm.com/developerworks/br/downloads/r/rft/`. Acessado: 10-03-2016.

[4] Marathon. `https://marathontesting.com/`. Acessado: 09-02-2016.

[5] Mercury tours. `http://newtours.demoaut.com/mercurywelcome.php/`. Acessado: 07-12-2015.

[6] Online store|toolsqa. `http://store.demoqa.com/`. Acessado: 07-12-2015.

[7] Selenium: Beginners guide. `http://book.theautomatedtester.co.uk/`. Acessado: 07-12-2015.

[8] Selenium projects. `http://docs.seleniumhq.org/`. Acessado: 21-01-2016.

[9] Serenity bdd. `http://thucydides.info/docs/serenity-staging/`. Acessado: 10-03-2016.

[10] Testng. `http://testng.org/doc/index.html`. Acessado: 19-01-2016.

[11] Why you can't compare cross browser execution times of selenium tests. `http://goo.gl/PcSC84`. Acessado: 07-04-2016.

[12] *Base de Conhecimento Em Teste de Software*. Martins Fontes, 2012.

[13] A. Arora. Extentreports. `http://extentreports.relevantcodes.com/`. Acessado: 19-01-2016.

[14] I. S. T. Q. B. *Certified Tester Foundation Level Syllabus*. ISTQB, 2011.

[15] K. Beck. *JUnit Pocket Guide*. O'Reilly Media, 2004.

[16] C. Beust and H. Suleiman. *Next Generation Java Testing: TestNG and Advanced Concepts*. Addison-Wesley Professional, 2007.

[17] R. Black. Investing in software testing: The cost of software quality. *Rex Black Consulting*, 2000.

[18] D. Burns. *Selenium 2 Testing Tools: Beginner's Guide*. Packt Publishing, 2012.

[19] M. Chrissis, M. Konrad, and S. Shrum. *CMMI: Guidelines for Process Integration and Product Improvement*. SEI series in software engineering. Addison-Wesley, 2003.

[20] M. Delamaro, J. Maldonado, and M. Jino. *Introdução ao teste de software*. CAMPUS - RJ, 2007.

[21] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[22] W. S. Humphrey. Characterizing the software process: a maturity framework. *Software, IEEE*, 5(2):73–79, 1988.

[23] ISO/IEC. *ISO/IEC 9126-1. Engenharia de software - Qualidade de produto*. ISO/IEC, 2003.

[24] C. Kaner, J. Bach, and B. Pettichord. *Lessons Learned In Software Testing*. Wiley India Pvt. Limited, 2008.

[25] mptBr. *Guia de Referência do Modelo – MPT.Br.* mptBr, 2013.

[26] G. Myers, C. Sandler, T. Badgett, and T. Thomas. *The Art of Software Testing.* Business Data Processing: A Wiley Series. Wiley, 2004.

[27] R. Pressman. *Engenharia de Software.* McGraw Hill Brasil, 2009.

[28] W. Randelshofer. Monte screen recorder. `http://www.randelshofer.ch/monte/`. Acessado: 19-01-2016.

[29] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in Software Engineering.* Computer Science. Springer Berlin Heidelberg, 2012.

APÊNDICE

A. LISTA DE MÉTODOS

Os métodos apresentado nessa seção, são os métodos criados para a prova de conceito e disponíveis para interação como *Mercury Tours*:

- `registerUser` - método utilizado para preencher os campos da tela de registro e realizar o registro de um usuário para acesso a aplicação, de acordo com os dados presentes na planilha;
- `login` - Método que preenche os campos, de acordo com os dados presentes na planilha e realiza o *login* de um usuário na aplicação(aciona o botão *submit*);
- `findFlight` - Método utilizado para informar os critérios e buscar um voo;
- `selectFlight` - Método responsável por preencher os campos da tela de seleção de voos;
- `passager0` - Método responsável por preencher os dados do primeiro passageiro, de acordo com os dados presentes na planilha;
- `passager1` - Método responsável por preencher os dados do segundo passageiro, de acordo com os dados presentes na planilha;
- `passager2` - Método responsável por preencher os dados do terceiro passageiro, de acordo com os dados presentes na planilha;
- `passager3` - Método responsável por preencher os dados do quarto passageiro, de acordo com os dados presentes na planilha;
- `bookFlightCreditCard` - Método que preenche as informações dos campos de cartão de crédito na tela de reserva de voos, de acordo com os dados presentes na planilha;
- `bookFlightAddress` - Método que preenche as informações do endereço de venda na tela de reserva de voos, de acordo com os dados presentes na planilha;
- `bookFlightDeliveryAddress` - Método que preenche as informações do endereço de entrega na tela de reserva de voos, de acordo com os dados presentes na planilha;
- `bookFlightBuy` - Método que efetiva a reserva, na tela de reserva de voos, de acordo com os dados presentes na planilha;
- `confirmFlight` - Método que verifica se a reserva foi efetuada com sucesso;
- `logoff` - Realiza o *logoff* na aplicação;
- `navigate` - Método que realiza a navegação entre todas as páginas disponíveis na aplicação;
- `vacationPage` - Método que navega até a página *Vacations*;
- `carPage` - Método que navega até a página *Car*;
- `hotelsPage` - Método que navega até a página *Hotels*;
- `contactPage` - Método que navega até a página *Contact*;
- `supportPage` - Método que navega até a página *Support*;
- `cruisePage` - Método que navega até a página *Cruise*.

B. CASOS DE USO MERCURY TOURS

Caso de uso: Reservar voo

1 INFORMAÇÕES GERAIS

1.1 Descrição do caso de uso

O objetivo desse caso de uso é apresentar a funcionalidade de reserva de voos no sistema *Mercury Tours*, descrevendo as interações e fluxos necessários para a reserva de voos por um usuário registrado no sistema.

2 Pré-Condições

2.1 Acesso ao *Mercury Tours*

É necessário que o ator possua acesso ao link do sistema.

2.2 Usuário cadastrado no *Mercury Tours*

É necessário que o ator possua um usuário válido para realizar *login* e reserva de passagens no sistema.

3 Cenários

3.1 Cenário Principal

O cenário começa quando o ator deseja realizar a reserva de voos no sistema.

1. O ator preenche as informações na tela inicial do sistema para *login*;
   1.1. O sistema permite o preenchimento dos campos "*User Name*" e "*Password*";
2. O ator aciona o botão "*Submit*" para acessar o sistema;
   2.1. O sistema autentica o usuário e apresenta a tela "*Flight Finder*" [5.1];
3. O ator seleciona os detalhes do voo e as suas preferências e aciona o botão "*Continue*" [5.1];
   3.1. O sistema apresenta a tela "*Select Flight*", exibindo os voos encontrados de acordo com os dados preenchidos no filtro anterior;
4. O ator seleciona o(s) voo(s) desejados(s) e aciona o botão "*Continue*";
   4.1. O sistema apresenta a tela "*Book a Flight*", exibindo as informações do(s) voo(s) selecionado(s) e o(s) dado(s) do(s) passageiro(s) para preenchimento [5.2];
5. O ator preenche as informações do(s) passageiro(s) e aciona o botão "*Secure Purchase*" [3.2.1] [5.2]
   5.1. O sistema registra as informações, efetiva a reserva do(s) voo(s) e apresenta a mensagem: *"Your itinerary has been booked! Please print a copy of this screen for your records. Thank you for choosing Mercury Tours"*

6. O ator realiza o *logoff* do sistema;

   6.1. O sistema efetiva o *logoff* e retorna a tela inicial;

7. Fim do caso de uso

3.2 Cenários Alternativos

3.2.1 Passagem fora do *United States*

Quando o ator informar um endereço de entrega fora dos Estados Unidos, o sistema deve apresentar o seguinte alerta:

"You have chosen a mailing location outside of the United States and its territories. An additional charge of $6.5 will be added as mailing charge."

4 Pós Condições

4.1 Reserva realizada com sucesso

O sistema deverá realizar a reserva de voo com sucesso de acordo com as informações fornecidas.

5 Tabelas

5.1 Campos da tela *Flight Finder*

Campos	Tipo	Valor	Descrição
		Flight Details	
Type	*Radio Button*	*Round Trip(default)* *One way*	Indica o tipo de viagem que será utilizada na busca por voos
Passengers	*Combo box*	1;2;3;4	Indica a quantidade de passageiros que será utilizada na busca por voos
Departing From	*Combo box*	Lista de locais de partida disponíveis	Indica qual o local de partida que será utilizado na busca por voos
On	Data	Calendário dia e mês	Indica qual o mês e o dia que será utilizado na busca por voos
Arriving in	*Combo box*	Lista de destinos disponíveis	Indica qual o destino que será utilizado na busca por voos
Returning	Data	Calendário dia e mês	Indica a data de retorno que será utilizada na busca por voos
		Preferences	
Service Class	*Radio Button*	Economy class; Business class; First class;	Indica a classe que será utilizada na busca por voos
Airline	*Combo box*	Lista com as companhias aéreas disponíveis	Indica qual a companhia aérea que será utilizada na busca por voos

5.2 Campos da tela *Book a Flight*

Campos	Tipo	Valor	Descrição
Passengers			
First Name	*Text*	-	Deve ser preenchido com o primeiro nome do passageiro
Last Name	*Text*	-	Deve ser preenchido com o segundo nome do passageiro
Meal	*Combo box*	Lista com refeições disponíveis no voo	Deve ser selecionada a preferência de refeição para o passageiro
Credit Card			
Card Type	Combo box	Bandeiras disponíveis	Deve ser selecionada a bandeira do cartão de crédito para compra
Number	Text	-	Deve ser preenchido com o número do cartão de crédito
Expiration	Combo box	-	Deve ser selecionado o mês e ano de expiração do cartão de crédito
First Name	Text	-	Deve ser informado o primeiro nome do titular do cartão de crédito
Middle	Text	-	Deve ser informado o nome do meio do titular do cartão de crédito
Last	Text	-	Deve ser informado o último nome do titular do cartão de crédito
Billing Address			
Address	*Text*	-	Deve ser preenchido com o endereço de cobrança do titular do cartão de crédito
City	*Text*	-	Deve ser preenchido com a cidade de cobrança do titular
State/Province	Text	-	Deve ser preenchido com o estado de cobrançado titular
Postal Code	Text	-	Deve ser preenchido com o código postal de cobrança do titular
Country	Combo box	Lista com os países disponíveis para cobrança	Deve ser preenchido com o país de cobrança do titular
Delivery Address			
Address	*Text*	-	Deve ser preenchido com o endereço de entrega do titular
City	*Text*	-	Deve ser preenchido com a cidade de entrega do titular
State/Province	Text	-	Deve ser preenchido com o estado de entrega do titular
Postal Code	Text	-	Deve ser preenchido com o código postal de entrega do titular
Country	Combo box	Lista com os países disponíveis	Deve ser preenchido com o país de entrega do titular

Caso de uso: Registrar Usuário

1 INFORMAÇÕES GERAIS

1.1 Descrição do caso de uso

O objetivo desse caso de uso é apresentar a funcionalidade de registro de usuário no sistema *Mercurt Tours*, descrevendo as interações e fluxos necessários para o cadastro de um usuário para utilização do sistema.

2 Pré-Condições

2.1 Acesso ao *Mercury Tours*

É necessário que o ator possua acesso ao link do sistema.

3 Cenários

3.1 Cenário Principal

O cenário começa quando o ator deseja realizar o registro de um usuário no sistema, para que ele possa realizar a reserva de voos no sistema.

1. O ator acessa o link "*Register*" na barra superior do sistema;
    1.1. O sistema apresenta a tela de registro de usuário com os campos disponíveis para preenchimento [5.1];
2. O ator preenche as informações do usuário que ele deseja registrar [5.1];
    2.1. O sistema permite o preenchimento dos campos;
3. O ator aciona o botão "*Submit*" para enviar os dados do usuário para registro;
    3.1. O sistema realiza o registro do usuário e apresenta a mensagem de confirmação do registro de usuário;
4. Fim do caso de uso

3.2 Cenários Alternativos

Não se aplica

4 Pós Condições

Devera ser descrito o estado do sistema após o fim do caso de uso

4.1 Usuário Registrado com sucesso

O sistema deve apresentar o usuário registrado e com permissão de acesso ao sistema *Mercury Tours*

5 Tabelas

5.1 Campos da tela ***Flight Finder***

Campos	Tipo	Valor	Descrição
Contact Information			
First Name	*Text*	-	Deve ser preenchido com o primeiro nome do usuário
Last Name	*Text*	-	Deve ser preenchido com o segundo nome do usuário
Phone	*Text*	-	Deve ser preenchido com o telefone do usuário
Email	Text	-	Deve ser preenchido com o e-mail do usuário
Mailing Information			
Address	*Text*	-	Deve ser preenchido com o endereço do usuário
City	*Text*	-	Deve ser preenchido com a cidadedo usuário
State/Province	Text	-	Deve ser preenchido com o estado do usuário
Postal Code	Text	-	Deve ser preenchido com o código postal do usuário
Country	Combo box	Lista com os países disponíveis	Deve ser preenchido com o país do usuário
User Information			
User Name	*Text*	-	Deve ser preenchido com o usuário que será utilizado para logar no sistema
Password	*Text*	-	Deve ser preenchido com o password do usuário
Confirm Password	*Text*	-	Deve ser preenchido com a confirmação do password do usuário