Desenvolvimento de *Frameworks* de Testes Automáticos de *Software*

Luís Guilherme Gomes Albuquerque dos Santos Castelo

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia

Orientador: Professor Doutor António Constantino Lopes Martins

Júri:

Presidente:

Professor Doutor Paulo Alexandre Fangueiro Oliveira Maio, Professor Adjunto, DEI/ISEP

Vogais:

Professor Doutor Ângelo Manuel Rego E Silva Martins, Professor Adjunto, DEI/ISEP

Professor Doutor António Constantino Lopes Martins, Professor Adjunto, DEI/ISEP

Porto, Outubro 2015

Aos meus pais, irmã e sobrinho

Resumo

O objetivo deste trabalho é o desenvolvimento de *frameworks* de testes automáticos de *software*. Este tipo de testes normalmente está associado ao modelo evolucionário e às metodologias ágeis de desenvolvimento de *software*, enquanto que os testes manuais estão relacionados com o modelo em cascata e as metodologias tradicionais. Como tal foi efetuado um estudo comparativo sobre os tipos de metodologias e de testes existentes, para decidir quais os que melhor se adequavam ao projeto e dar resposta à questão "Será que realmente compensa realizar testes (automáticos)?".

Finalizado o estudo foram desenvolvidas duas *frameworks*, a primeira para a implementação de testes funcionais e unitários sem dependências a ser utilizada pelos estagiários curriculares da *LabOrders*, e a segunda para a implementação de testes unitários com dependências externas de base de dados e serviços, a ser utilizada pelos funcionários da empresa.

Nas últimas duas décadas as metodologias ágeis de desenvolvimento de *software* não pararam de evoluir, no entanto as ferramentas de automação não conseguiram acompanhar este progresso. Muitas áreas não são abrangidas pelos testes e por isso alguns têm de ser feitos manualmente. Posto isto foram criadas várias funcionalidades inovadoras para aumentar a cobertura dos testes e tornar as *frameworks* o mais intuitivas possível, nomeadamente:

1. *Download* automático de ficheiros através do *Internet Explorer 9* (e versões mais recentes).

2. Análise do conteúdo de ficheiros .pdf (através dos testes).

3. Obtenção de elementos *web* e respetivos atributos através de código *jQuery* utilizando a *API WebDriver* com *PHP bindings*.

4. Exibição de mensagens de erro personalizadas quando não é possível encontrar um determinado elemento.

As *frameworks* implementadas estão também preparadas para a criação de outros testes (de carga, integração, regressão) que possam vir a ser necessários no futuro. Foram testadas em contexto de trabalho pelos colaboradores e clientes da empresa onde foi realizado o projeto de mestrado e os resultados permitiram concluir que a adoção de uma metodologia de desenvolvimento de *software* com testes automáticos pode aumentar a produtividade, reduzir as falhas e potenciar o cumprimento de orçamentos e prazos dos projetos das organizações.

Palavras-chave: metodologias ágeis, *frameworks*, testes automáticos, funcionais, unitários

Abstract

The objective of this work is the development of automated *software* testing frameworks. This type of tests is normally associated with the evolutionary model and agile *software* development methodologies, while the manual tests are associated with the waterfall model and traditional methodologies. Therefore a comparative study of the existing methodology and test types was performed to decide which ones were best suited to the project and to answer the question "Is it really worth it to perform (automatic) tests?".

After completing the study two frameworks were developed, the first for the implementation of functional and unit tests without dependencies to be used by LabOrders' temporary trainees, and the second for the implementation of unit tests with external database and service dependencies to be used by the company's employees.

In the last two decades the agile *software* methodologies haven't stopped evolving, however the automation tools haven't been able to accompany this progress. Many areas aren't covered by the tests so some of them have to be done manually. Given this situation, several innovative features have been created to increase test coverage and make the frameworks as intuitive as possible, namely:

1. Automatic file download in Internet Explorer 9 (and later versions).
2. .pdf file content analysis (through tests).
3. Obtaining web elements and respective attributes via jQuery code using the WebDriver API with PHP bindings.
4. Displaying custom error messages when an element can't be found.

The implemented frameworks are also ready for the creation of other tests (load, integration, regression) that may be needed in the future. They were tested in work context by the employees and customers of the company where the master's project was conducted and the results showed that the adoption of a *software* development methodology with automatic testing can increase productivity, reduce bugs and improve compliance with the budgets and deadlines of the organization projects.

Keywords: agile methodologies, frameworks, automated tests, functional, unit

Agradecimentos

Esta tese representa o culminar de um percurso académico de 17 anos e só foi possível graças à contribuição de um grupo de pessoas a quem gostaria de agradecer.

Aos meus pais, pela sua convergência de esforços, quer a nível material quer a nível emocional, bem como a sua dedicação, compreensão, motivação, força, ajuda e apoio incondicional.

À minha irmã e ao meu sobrinho, pelas palavras de encorajamento e incentivo.

Ao meu orientador, Doutor António Constantino Lopes Martins, pela sua disponibilidade, aconselhamento e revisão deste documento.

Ao Instituto Superior de Engenharia do Porto (ISEP), pela qualidade de ensino prestada.

À *LabOrders*, pela oportunidade de realizar o projeto de mestrado nas suas instalações.

Aos meus colegas, pelos bons momentos que passei na sua companhia.

Obrigado a todos!

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos

Lista de Acrónimos

AJAX	*Asynchronous JavaScript And XML*
API	*Application Program Interface*
ATLM	*Automated Test Lifecycle Metodology*
C3	*Chrysler Comprehensive Compensation System*
CAPTCHA	*Completely Automated Public Turing Test to tell Computers and Humans Apart*
CMS	*Content Management System*
CSS	*Cascading Style Sheets*
CSV	*Comma-Separated Values*
ENEB	Encontro Nacional de Estudantes de Biologia
FDD	*Feature Driven Development*
FTP	*File Transfer Protocol*
GraPE	Graduados Portugueses no Estrangeiro
GUI	*Graphical User Interface*
HP	*Hewlett-Packard*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
iBET	Instituto de Biologia Experimental e Tecnológica
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronic Engineers*
IGC	Instituto Calouste Gulbenkian
ISEP	Instituto Superior de Engenharia do Porto
MVC	*Model-View-Controller*

NPM	*Node Package Manager*
ORM	*Object-Relational Mapping*
PAT	Prova de Aptidão Tecnológica
PEAR	*PHP Extension and Application Repository*
PHP	*Hypertext Preprocessor*
PI	*Principal Investigator*
PME	Pequenas e Médias Empresas
QTP	*QuickTest Professional*
SADEC	Sistemas de Apoio à Decisão
SAP	*Systems Applications and Products*
SMTP	*Simple Mail Transfer Protocol*
SKU	*Stock Keeping Unit*
td	*table data*
TDD	*Test-Driven Development*
TFD	*Test-First Development*
tr	*table row*
UFT	*Unified Functional Testing*
W3C	*World Wide Web Consortium*
WAMP	*Windows/Apache/MySQL/PHP*
WWW	*World Wide Web*
XML	*eXtensible Markup Language*
XP	*eXtreme Programming*

1 Introdução

As ferramentas de testes automáticos de *software* têm evoluído ao longo do tempo juntamente com as metodologias de desenvolvimento de *software*, se bem que a ritmos diferentes porque só após o surgimento das metodologias ágeis é que os testes automáticos foram amplamente adotados. Hoje em dia cada vez mais empresas/programadores seguem metodologias ágeis, que aliás já ultrapassaram as metodologias tradicionais em número de utilizadores [Standish Group, 1995] [Version One, 2013]. Normalmente o desenvolvimento de *software* sem seguir nenhuma metodologia é associado à ausência de testes. Os testes manuais são associados às metodologias tradicionais e os testes automáticos são associados às metodologias ágeis. Isto denota o facto da evolução das *frameworks* de testes automáticos estar relacionada com o progresso do desenvolvimento evolucionário.

O futuro adivinha-se prometedor para a área dos testes automáticos de *software* com as metodologias ágeis tão em voga, e o corrente projeto pretende prestar o seu contributo dando continuidade a este progresso, incentivando a disseminação do desenvolvimento evolucionário e criando funcionalidades inovadoras que não se encontram disponíveis nas ferramentas que foram analisadas no âmbito deste documento.

1.1 Motivação

A motivação para a escolha do tema deveu-se a vários fatores. Após ter concluído a licenciatura o autor da tese tomou a decisão de adiar a entrada no mercado de trabalho para se dedicar a tempo inteiro ao mestrado com o objetivo de o concluir em tempo útil e com a melhor nota possível. Foi selecionada a proposta disponível que mais se aproximava do ramo de Sistemas Gráficos e Multimédia, já que o desenvolvimento de uma *framework* de testes automáticos envolve o uso de vários meios de comunicação em simultâneo para transmissão de dados estáticos (texto, fotografia, gráfico) e dinâmicos (vídeo, áudio, animação). Os testes automáticos são também uma das áreas de que o autor mais gosta e em que tem mais experiência visto que desenvolveu uma Prova de Aptidão Tecnológica (PAT) e o projeto final de licenciatura com muitas das tecnologias que irá utilizar no âmbito deste projeto. Para além disso a empresa escolhida para a realização da tese encontra-se incubada no Parque de Ciência e Tecnologia da Universidade do Porto (UPTEC), o que será útil para analisar o contexto de trabalho neste pólo de desenvolvimento e estabelecer contactos com várias organizações.

Foi então feito um estágio na empresa *LabOrders*, cujo principal produto consiste numa plataforma *online* de gestão de encomendas de material de laboratório, para realização da tese de mestrado com o tema **"Desenvolvimento de *Frameworks* de Testes Automáticos de *Software*"**.

1.2 Objetivos Propostos

Esta dissertação pretende detalhar e estabelecer a distinção entre metodologias de desenvolvimento de *software* tradicionais e ágeis, identificar as suas atividades-chave comuns e reconhecer a validação de *software* como uma dessas atividades. Tenciona igualmente demonstrar com o suporte de dados estatísticos que as metodologias ágeis são cada vez mais utilizadas e refletem um aumento de produtividade e redução de falhas, bem como o cumprimento dos prazos e orçamentos. Irá ser realizada uma análise à metodologia de desenvolvimento de *software* da *LabOrders* com o propósito de apresentar sugestões de melhoria e irão também ser identificadas estratégias para potenciar os benefícios e contornar os obstáculos/dificuldades esperados da adoção deste novo processo. As decisões respetivas serão tomadas tendo como base as vantagens e desvantagens do desenvolvimento evolucionário face ao sequencial e dos testes automáticos de *software* face aos manuais. Será ainda importante pormenorizar e determinar a diferença entre testes de caixa-preta e testes de caixa-branca, bem como entre testes funcionais e unitários, para que a codificação dos mesmos seja o mais eficaz e eficiente possível seguindo as práticas recomendadas. Finalmente será ressalvada a importância da escolha de ferramentas de testes automáticos e efetuado um estudo comparativo para selecionar as mais apropriadas ao desenvolvimento de um projeto completo e inovador que permita à empresa seguir a metodologia sugerida, bem como efetuar o processo de garantia da qualidade de todos os seus produtos.

Os objetivos propostos pela *LabOrders* para a tese podem ser sintetizados nos seguintes pontos.

Objetivos gerais:

- Dar resposta à questão proposta "Será que realmente compensa realizar testes (automáticos)?".
- Automatizar o processo de garantia da qualidade da *LabOrders* de forma a torná-lo mais objetivo, eficaz, eficiente, rápido, fiável e com menos erros.
- Assegurar através dos testes automáticos que todo o trabalho desenvolvido pela empresa se encontra a funcionar corretamente.

Objetivos específicos:

- Realizar um estudo do estado da arte com o intuito de identificar a metodologia de desenvolvimento de *software* e as ferramentas de testes automáticos mais adequadas à empresa.
- Desenvolver uma *framework* de testes funcionais e unitários sem dependências externas.
- Desenvolver uma *framework* de testes unitários com dependências externas (base de dados e serviços).

- Garantir que as *frameworks* funcionam com os três *browsers* utilizados pela empresa: *Firefox*, *Chrome* e *Internet Explorer*.
- Codificar testes demonstrativos das novas funcionalidades implementadas.
- Codificar testes funcionais não demonstrativos das novas funcionalidades implementadas (utilizando funcionalidades nativas das ferramentas escolhidas ou que foram desenvolvidas antes do projeto de mestrado ter tido início).
- Elaborar uma aplicação de testes como prova de conceito.
- Criar um novo módulo de testes unitários para a *framework Kohana* utilizada pela *LabOrders* para o desenvolvimento da sua plataforma *online* de encomendas.

Requisitos e funcionalidades:

- *Download* automático de ficheiros através do *Internet Explorer 9* (e versões mais recentes).
- Análise do conteúdo de ficheiros .pdf (através dos testes).
- Obtenção de elementos *web* e respetivos atributos através de código *jQuery*.
- Exibição de mensagens de erro personalizadas quando não é possível encontrar um determinado elemento.
- Correção da funcionalidade de envio do relatório de testes por correio eletrónico desenvolvida pela empresa antes do projeto de mestrado ter tido início.
- Desenvolvimento de uma aplicação de testes como prova de conceito.
- Criação de um novo módulo de testes unitários para a *framework Kohana* utilizada pela *LabOrders* para o desenvolvimento da sua plataforma *online* de encomendas.

1.3 Planificação e Metodologias

A planificação do projeto teve início com o levantamento dos requisitos junto da *LabOrders*. De seguida foi realizado um estudo do estado da arte que permitiu:

- Alcançar um entendimento amplo sobre os domínios em que a tese se insere.
- Apurar qual a metodologia de *software* mais adequada à empresa.
- Obter dados estatísticos a partir de vários relatórios/pesquisas de diferentes instituições para fundamentar a resposta à questão proposta: "Será que realmente compensa realizar testes (automáticos)?".
- Selecionar as ferramentas e as abordagens aos testes funcionais e unitários mais indicadas para utilizar no projeto.
- Conseguir a informação para a configuração e desenvolvimento das *frameworks* de testes automáticos.

Por fim as novas funcionalidades identificadas na fase de levantamento de requisitos foram criadas e testadas, o que assegurou e demonstrou a sua correta implementação.

A metodologia de investigação seguida compreende as seguintes técnicas de recolha de dados:

- Revisão da literatura sobre o tema da tese.
- Pesquisa de artigos científicos e páginas *web* relacionados com o assunto do trabalho de mestrado.

As referências mais relevantes, válidas, rigorosas, interessantes e fiáveis foram escolhidas tendo como base a experiência e contacto direto do autor com o objeto de estudo, bem como o aconselhamento junto de especialistas e colegas.

A metodologia de desenvolvimento utilizada foi a Programação Extrema (em Inglês *eXtreme Programming* ou XP) que irá ser detalhada posteriormente no capítulo **"Metodologias de Desenvolvimento e Testes Automáticos de *Software*"**.

1.4 Estrutura do Documento

Este documento está dividido em 5 capítulos.

O 1º capítulo é a **"Introdução"** que contém uma descrição breve do trabalho desenvolvido, a motivação para a escolha do tema, os objetivos propostos pela empresa, um resumo do estado da arte nos domínios em que a tese se insere, o contributo para a área de estudo em questão e a planificação e metodologias seguidas no projeto, bem como a estrutura da dissertação.

O 2º capítulo intitulado **"Metodologias de Desenvolvimento e Testes Automáticos de *Software*"** corresponde ao estado da arte e aborda aspetos como as metodologias de desenvolvimento de *software* ágeis e tradicionais, o Modelo Clássico, vários exemplos detalhados de metodologias ágeis (*Scrum*, *Crystal*, Desenvolvimento Guiado por Funcionalidades, Programação Extrema), as vantagens do modelo evolucionário, o desenvolvimento guiado por testes, a refatorização, os testes de caixa-preta, caixa-branca, funcionais e unitários, o estudo comparativo de ferramentas de automação e a Metodologia do Ciclo de Vida dos Testes Automáticos.

O 3º capítulo é a **"Descrição Técnica"** que engloba a arquitetura e a estrutura das *frameworks*, a criação das novas funcionalidades, da aplicação de testes e do novo módulo de testes unitários, a correção de funcionalidades já existentes desenvolvidas pela *LabOrders* antes do projeto de mestrado ter tido início e o desenvolvimento de testes funcionais.

O 4º capítulo é a **"Avaliação"** e descreve a forma como foram analisados os resultados do projeto, tanto pelos colaboradores da empresa como pelos clientes.

O 5º capítulo é a **"Conclusão"** que abrange o balanço final, os aspetos principais do trabalho realizado, limitações e possível desenvolvimento futuro.

Após o último capítulo encontram-se as **"Referências"** e os **"Anexos"** com a configuração de ambas as *frameworks*: a primeira de testes funcionais e unitários sem dependências externas (**"Configuração do *Selenium WebDriver*, *PHPUnit*, *NetBeans* e WAMP"**) e a segunda de testes unitários com dependências externas (**"Configuração do *Kohana*, *SimpleTest* e *Phactory*"**). Inclui também a **"Configuração dos *Browsers*"**, excertos de código dos testes/casos de teste e a página principal de um dos *sites WordPress* criados.

2 Metodologias de Desenvolvimento e Testes Automáticos de *Software*

Foi efetuado um estudo do estado da arte sobre o desenvolvimento ágil e tradicional de *software* para averiguar qual a metodologia de desenvolvimento e estilo de programação que deveriam ser adotados na *LabOrders*. Entende-se por estilo de programação as regras relativamente a nomenclatura de variáveis e constantes, escrita de classes, funções e comentários. Foram também identificados os benefícios esperados e os obstáculos/dificuldades que podem surgir da adoção destas práticas, bem como as razões, vantagens e desvantagens de automatizar o teste de *software*. Foi igualmente estabelecida a diferença entre as categorias de testes automáticos em que se enquadram os testes funcionais e unitários.

Finalmente foi sublinhada a importância da escolha das ferramentas de testes automáticos e realizado um comparativo entre diversas ferramentas de testes funcionais e unitários, juntamente com a respetiva justificação das opções tomadas.

2.1 Metodologias de Desenvolvimento de *Software*

Uma metodologia de desenvolvimento ou processo de *software* é "um conjunto de atividades e resultados associados que auxiliam na produção de *software"* [Sommerville, 1995]. A adoção destas práticas é essencial para a produtividade de uma empresa na medida em que reduzem os prazos, custos de desenvolvimento e erros, aumentando a qualidade do produto final.

Apesar de existirem várias metodologias de desenvolvimento de *software*, as atividades principais são comuns a todas elas. Essas atividades fundamentais foram identificadas por Sommerville (1995) e são as seguintes:

- **Especificação**: levantamento dos requisitos funcionais (funções do sistema) e não-funcionais (desempenho, usabilidade, confiabilidade, disponibilidade, segurança, complexidade, compreensibilidade, manutenção, entre outros) junto do cliente.
- **Projeto e implementação**:
    - Elaboração da documentação, nomeadamente diagramas de casos de uso, classes e atividades, modelo de dados, fluxogramas, entre outros.
    - Implementação do *software* de acordo com os requisitos recolhidos durante a especificação.
- **Validação de *software**:*** codificação e execução de testes (manuais ou automáticos: funcionais, unitários, de integração, de carga, de regressão, entre outros) para garantia da qualidade do *software*.

- **Evolução**: capacidade de evoluir o *software* de acordo com as necessidades futuras do cliente.

Existem duas classificações de metodologias de desenvolvimento de *software* [Soares, 2004]: as tradicionais, também conhecidas como pesadas, que são orientadas a documentação e planeamento, e as ágeis, que têm como objetivo o desenvolvimento rápido de *software* com qualidade.

2.1.1 Metodologias Tradicionais

As metodologias tradicionais surgiram nos anos 70 [Semedo, 2012], uma época em que o desenvolvimento de *software* era realizado através de terminais sem qualquer capacidade de processamento e que apenas enviavam os comandos digitados pelo utilizador para uma *mainframe* (computador de alta *performance* normalmente destinado ao processamento de grandes volumes de dados), de forma a serem processados. Como tal, o custo de modificar o *software* era muito elevado porque o acesso aos computadores era limitado e não existiam ferramentas de apoio aos programadores como depuradores (em Inglês *debuggers)* e analisadores de código. Entende-se por depurador um programa de computador que permite obter um nível de controlo superior sobre a execução de outros programas [Techopedia]. Possibilita correr programas instrução a instrução e sob determinadas condições, parar a execução em pontos específicos, modificar o estado dos programas enquanto estão a correr e examinar o conteúdo das variáveis. Por sua vez analisador de código é a ferramenta que detecta e realça erros de lógica e sintaxe. Isto resultou na necessidade de planear e documentar o *software* antes da sua implementação.

A principal metodologia tradicional, que ainda é muito utilizada nos dias de hoje, é o Modelo Clássico, também conhecido como Sequencial ou em Cascata [Soares, 2004].

2.1.1.1 Modelo Clássico

O Modelo Clássico criado por Royce em 1970, também conhecido como abordagem *top-down*, foi a primeira metodologia de desenvolvimento de *software* a ser publicada [Royce, 1970]. Este modelo compreende uma sequência de atividades em que cada uma possui documentação própria e deve ser aprovada pelo cliente antes de se dar início à atividade seguinte, sendo este o motivo do Modelo Clássico também ser conhecido como Sequencial ou em Cascata.

Geralmente incluem-se nas atividades do Modelo Clássico:

- A **definição de requisitos**: Royce deu o nome de definição de requisitos à especificação (nomenclatura de Sommerville) mas são a mesma atividade. A definição de requisitos já foi abordada anteriormente no capítulo **"Metodologias de Desenvolvimento de *Software*"**, consultar o texto do tópico **"Especificação"**.
- O **projeto do *software***: consultar o tópico **"Projeto e implementação"** do capítulo anterior: **"Metodologias de Desenvolvimento de *Software*"**.

- A **implementação e testes unitários**: em relação à implementação consultar o texto do tópico **"Projeto e implementação"** do capítulo anterior: **"Metodologias de Desenvolvimento de *Software*"**. Os testes unitários enquadram-se na validação de *software* que também já foi abordada anteriormente no capítulo supracitado, consultar o texto do tópico **"Validação de *software*"**.
- A **integração e teste**: esta atividade também se enquadra na validação de *software* que já foi abordada no capítulo anterior **"Metodologias de Desenvolvimento de *Software*"**, consultar o tópico **"Validação de *software*"**.
- A **operação e manutenção**: A operação é uma atividade que tem início após a aceitação provisória do *software* por parte do cliente e consiste em colocar o sistema em funcionamento no ambiente para o qual foi projetado. Por sua vez a manutenção é o processo de melhoria e otimização do *software* desenvolvido que consiste na correcção de erros detetados durante a utilização do cliente e na criação de novas funcionalidades.

A vantagem do Modelo Clássico face à não adoção de nenhuma metodologia de desenvolvimento de *software* é o facto das atividades serem validadas pelo cliente, permitindo a redução dos erros e respetivos riscos associados [Soares, 2004]. No entanto esta é também a sua grande desvantagem em relação ao desenvolvimento ágil porque o projeto é dividido rigorosamente em diferentes atividades e cada uma delas deve ser concluída antes de se avançar para a seguinte. Isto dificulta a modificação e evolução do projeto visto que uma alteração numa atividade implica alterações nas atividades posteriores. Por exemplo, a adição de um requisito após a fase de planeamento resulta na modificação do diagrama de casos de uso e possivelmente do diagrama de classes, modelo de dados, fluxogramas, testes e pode levar à necessidade de refatorizar o código, entre outras tarefas. Como tal apenas deve ser utilizado quando os requisitos estiverem bem definidos e forem bem compreendidos.

A **Figura 1** ilustra graficamente o Modelo Clássico [Soares, 2004]. Numa primeira fase é feito o levantamento dos requisitos do *software* junto do cliente. De seguida realiza-se o projeto do *software* onde é criada toda a documentação associada. Na fase de implementação e teste de unidades o *software* é desenvolvido de acordo com os requisitos recolhidos e são codificados os testes unitários de forma a garantir que todas as funcionalidades foram implementadas corretamente. Na fase de integração e teste do sistema são reunidos todos os subcomponentes desenvolvidos num único sistema que após testado pelo cliente, deve ser corrigido caso tenha sido detetada alguma anomalia, e aprovado provisoriamente. Finalmente, é configurado o ambiente de desenvolvimento do *software* a que se deve dar suporte ao longo do tempo, corrigindo eventuais erros que surjam no decorrer da sua utilização ou até mesmo desenvolvendo novas funcionalidades.

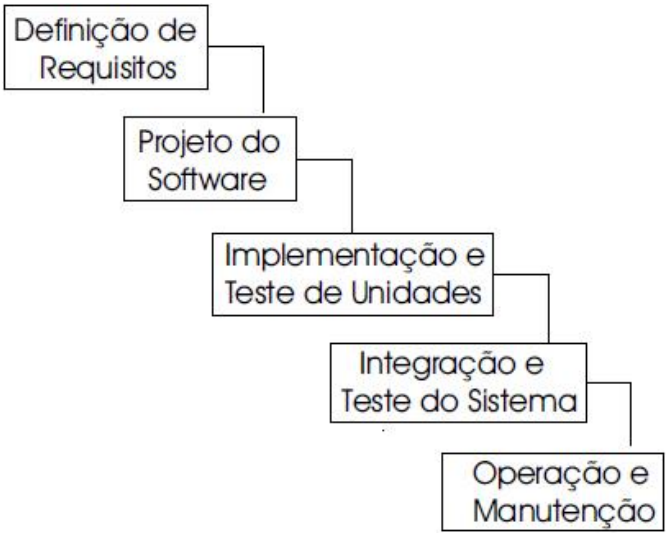

Figura 1- Modelo Clássico [Soares, 2004]

Os investigadores do desenvolvimento de *software* identificaram vários problemas associados ao Modelo Clássico e por conseguinte desencorajaram o seu uso. Fred Brooks afirmou no seu artigo *"No Silver Bullet: Essence and Accidents of Software Engineering"* que "a ideia de especificar totalmente um *software* antes do início da sua implementação é impossível", porque os seus requisitos são mutáveis e vão surgindo ao longo do ciclo de desenvolvimento [Brooks, 1987].

Tom Gilb refere que "o Modelo Clássico apresenta maiores riscos e menores possibilidades de sucesso em *softwares* de grande dimensão". Ao contrário do desenvolvimento incremental não adota uma perspetiva evolucionária em que podem ser adicionados e modificados requisitos em qualquer fase do projeto consoante o entendimento do cliente, logo não salvaguarda mal-entendidos resultantes de más interpretações ou de uma visão demasiado simplista do problema por parte deste [Gilb, 1988].

O motivo comum a todos estes problemas é o facto da alteração dos requisitos ao longo do ciclo de desenvolvimento obrigar a modificações em várias fases do projeto, o que pode ser resolvido através da adoção de metodologias de desenvolvimento de *software* ágeis, que irão ser abordadas em detalhe no próximo capítulo.

2.1.2 Metodologias Ágeis

O termo metodologias ágeis tornou-se popular em 2001 [Soares, 2004] quando vários especialistas em metodologias de desenvolvimento de *software* incluindo Ken Schwaber, Mike Beedle, Kent Beck, Martin Fowler, Robert C. Martin, Alistair Cockburn e Dave Thomas apresentaram, entre outras, as seguintes metodologias:

- **Programação Extrema**: metodologia baseada em comportamentos e atitudes que tem como principais objetivos o cumprimento dos prazos e orçamentos, a qualidade do produto final e a satisfação do cliente [Beck, 1999]. A XP impõe práticas de engenharia como o Desenvolvimento Guiado por Testes (em Inglês *Test-Driven Development* ou TDD) e as suas iterações duram uma ou duas semanas. Permite alterações dentro das iterações (podem ser adicionadas ou removidas tarefas nas iterações planeadas) e as equipas trabalham sob uma ordem rigorosa de prioridades definidas pelo cliente [Cohn, 2007].
- ***Scrum***: o nome *Scrum* é um termo desportivo proveniente do *Rugby* que é utilizado como metáfora para refletir o grau de cooperação necessário para ter sucesso [Harper, 2012]. Trata-se de uma metodologia inspirada pelos defeitos do Modelo Clássico do qual se pretende distanciar completamente, enfatizando a colaboração, *software* funcional, autogestão das equipas e a flexibilidade para se adaptar a realidades de negócio emergentes [Schwaber e Beedle, 2002] [James, 2009]. Distingue-se da XP porque tem iterações (denominadas *sprints*) mais longas, com duração entre duas semanas e um mês, não permite alterações nos seus *sprints* nem impõe a adoção de práticas de engenharia como o TDD. Tal como na XP o cliente (designado *Product Owner* no *Scrum*) estabelece as prioridades das tarefas mas cabe à equipa decidir a sequência em que as vão desenvolver [Cohn, 2007].
- ***Crystal***: família de metodologias adaptáveis, ultra-leves (evitam processos rigorosos e rígidos encontrados noutras metodologias) [Mauch, Bassuday et al, 2012] e impulsionadas pelas pessoas [Cockburn, 2009]. O nome *Crystal* provém dos cristais e pedras preciosas. O cristal é uma metáfora da família de metodologias e cada uma das suas faces representa "uma perspetiva diferente sobre o núcleo implícito de princípios e valores de cada metodologia", ou, em termos de *software*, "as técnicas, ferramentas, padrões e funções para cada metodologia" [Mauch, Bassuday et al, 2012].

  O criador do *Crystal*, Alistair Cockburn, considera a XP como "um membro honorário da família *Crystal*" devido às suas semelhanças. No entanto existe uma grande diferença que se deve ao facto do *Crystal* ser mais indicado para projetos de 2 a 8 pessoas já que é uma tentativa de descrever a metodologia mais leve e simples possível que ainda assim seja capaz de produzir bons resultados. Como tal não é adequada para projetos de maior envergadura, sendo nestes casos preferível uma metodologia mais disciplinada como a XP.

"A XP é muito mais disciplinada, o Crystal é muito mais tolerante, até com uma aparência desleixada. Diria que a XP é mais produtiva, o Crystal mais provável de ser seguido"
(Alistair Cockburn)

No entanto a XP consegue ser mais leve do que o próprio *Crystal* em alguns produtos de trabalho intermédios [Cockburn, 2011].

- **Desenvolvimento Guiado por Funcionalidades (em Inglês *Feature Driven Development* ou FDD)**: metodologia pragmática centrada no cliente e na arquitetura do *software* [Ambler]. Distingue-se da XP principalmente por ser mais escalável para equipas maiores, iniciar o processo de implementação pelo *design*, ter propriedade de código (cada classe pertence a uma única pessoa), encorajar inspecções formais e testes de regressão (que ocorrem sempre que o sistema é alterado) automatizados e ter as funções de programador-chefe e arquiteto de *software*-chefe.

O Manifesto Ágil foi escrito numa estância de esqui do Utah em 2001 e é um conjunto de princípios que devem ser adotados no desenvolvimento ágil de *software*. Para esse efeito foram estabelecidas atividades chave comuns a todas essas metodologias. A elaboração do manifesto resultou na criação da Aliança Ágil que representa "uma organização sem fins lucrativos com membros de todo o mundo, comprometida com o avanço dos princípios e práticas de desenvolvimento Ágil" [Agile Alliance].

Os quatro pontos principais do Manifesto Ágil são:

- **Os indivíduos e interações** devem ter prioridade sobre os processos e ferramentas: o maior ativo de qualquer empresa são as pessoas que a constituem e a forma como comunicam entre si. É mais importante promover um bom relacionamento e o bem-estar no local de trabalho do que impor a adoção de determinados processos e ferramentas, devendo-se sempre que possível tentar atingir consensos e evitar conflitos.
- O ***software* executável** deve ter prioridade sobre a documentação: é preferível apresentar uma versão funcional do *software* no final de cada iteração de forma ao cliente a poder avaliar do que apresentar-lhe documentação que possivelmente não compreenderá.
- A **colaboração do cliente** com prioridade sobre a negociação de contratos: é mais importante ter o cliente a colaborar no projeto esclarecendo dúvidas, validando e aprovando a documentação produzida, avaliando as versões de testes, entre outros aspetos, do que perder demasiado tempo a negociar com ele termos contratuais. Esse tempo pode ser melhor aproveitado resultando na redução dos prazos e custos de produção.
- As **respostas rápidas às mudanças** com prioridade sobre o planeamento: é impossível prever todos os cenários e requisitos através de planeamento sendo preferível estar preparado para dar resposta rápida a eventuais imprevistos que surjam durante o ciclo de desenvolvimento do *software* [Agile Manifesto, 2004].

Convém esclarecer que o Manifesto Ágil também dá valor aos processos e ferramentas, à documentação, à negociação de contratos e ao planeamento, no entanto atribui uma maior importância aos seus quatro pontos principais que se aproximam mais da forma de trabalhar das pequenas e médias empresas. Em casos de conflito deve-se optar sempre por seguir os quatro pontos principais do Manifesto Ágil em detrimento do resto.

2.1.3 Vantagens das Metodologias Ágeis Face às Metodologias Tradicionais

Muitas pequenas e médias empresas (PME) criam *software* sem uma metodologia bem definida porque o desenvolvimento tradicional não se adequa à sua forma de trabalhar [Soares, 2004]. Não têm recursos suficientes para adotar metodologias orientadas a documentação, com menos de 10 trabalhadores é impossível ter várias equipas especializadas em diferentes áreas: testes, *design*, desenvolvimento, entre outras, todos fazem de tudo um pouco. O escalonamento de tarefas é realizado na hora, à medida que vão surgindo e conforme a sua prioridade imediata. Como tal, as metodologias de desenvolvimento tradicional não se ajustam a esta realidade porque não permitem a alternância entre projetos, antes de se avançar para a próxima tarefa de um projeto é necessário concluir e documentar devidamente a tarefa atual, que tem de ser validada e aprovada pelo cliente. A não adoção de uma metodologia concreta resulta em *software* de baixa qualidade e difícil de modificar, que muitas vezes é entregue para além dos prazos estipulados e com um custo superior ao estabelecido. As metodologias ágeis resolvem este problema porque consideram que os requisitos são mutáveis e suportam desenvolvimento evolucionário, ou seja, é possível adicionar e modificar requisitos em qualquer fase do ciclo de desenvolvimento do *software*.

A consultora *Standish Group* publica anualmente um relatório chamado *CHAOS* [Standish Group, 1995] sobre o estado da agilidade. Segundo a interpretação do autor desta tese o nome *CHAOS* deve-se à razão do *Standish Group* considerar que o desenvolvimento de *software* seguindo metodologias tradicionais é um caos (Português para *chaos*), de onde surge também a motivação para publicar este relatório de forma a evidenciar e comprovar com números as vantagens do desenvolvimento ágil relativamente ao tradicional.

A 2ª edição, publicada em 1995 [Standish Group, 1995] utilizando 8380 projetos como base, mostra que "apenas 16,2% dos projetos foram entregues respeitando os prazos e os custos com todas as funcionalidades especificadas".

Aproximadamente 31 em cada 100 projetos foram cancelados antes de estarem completos e 52,7% foram entregues com menos funcionalidades, atrasos de 222% e média de custo 189% superior ao que foi especificado no início do projeto. Considerando todos os projetos que foram entregues após o prazo e com custos superiores, em média, apenas 61% das funcionalidades previstas originalmente foram implementadas. Mesmo os projetos cuja entrega é feita respeitando os limites de prazo e custo são de qualidade duvidosa, uma vez que os programadores provavelmente trabalharam pressionados e sob stress para entregarem o produto em tempo útil, o que segundo a mesma pesquisa pode quadruplicar o

número de erros. O Modelo Clássico é apontado como o principal motivo destas falhas. A recomendação final foi que o desenvolvimento de *software* deveria ser baseado em modelos evolucionários, o que poderia evitar muitas dos erros reportados [Soares, 2004].

Em comparação, a 17ª edição do relatório *CHAOS*, publicada em 2011 mostra uma evolução: "a percentagem de sucesso dos projetos ágeis foi de 42% contra 14% dos tradicionais e os projetos ágeis falharam em 9% contra 29% dos projetos em cascata". Entende-se por projetos tradicionais ou em cascata aqueles que utilizam metodologias de desenvolvimento tradicionais. Por sua vez projetos ágeis são aqueles que utilizam metodologias de desenvolvimento ágeis.

A *V1 Limited*, fabricante de sistemas eletrónicos de gestão de documentos, também realiza uma pesquisa anual sobre o estado da agilidade. A 7ª edição, publicada em 2013 [Version One, 2013], mostra que "84% das empresas que responderam praticam o desenvolvimento agil". Esta esmagadora percentagem e o aumento do número de projetos bem-sucedidos devem-se ao facto dos conceitos de garantia da qualidade terem mudado, antigamente o *software* era testado depois de desenvolvido e atualmente é testado ao longo de todo o ciclo de desenvolvimento. Segundo Deming, "a qualidade não deve ter origem na inspeção mas na melhoria do processo".

O desenvolvimento ágil implica que o código seja testado continuamente à medida que vai sendo escrito, garantindo a qualidade das partes e não apenas do produto final. Também favorece a inovação visto que existe uma maior margem de erro e os programadores têm mais liberdade para experimentar e pensar "fora da caixa", isto é, obter uma perspetiva diferente das ideias convencionais e apresentar soluções criativas.

Pode-se observar na **Tabela 1** um resumo das vantagens das metodologias ágeis face às tradicionais.

Tabela 1 – Comparação entre metodologias ágeis e tradicionais

	Metodologias ágeis	**Metodologias tradicionais**
Prazos	Mais curtos	Mais longos
Custo	Mais barato	Mais caro
Comunicação/*Feedback*	Constante	Inconstante
Simplicidade	Código mais simples	Código mais complexo
Fiabilidade	Menos erros	Mais erros
Evolução	Sim	Não
Aprovação do cliente	Ao longo de todo o projeto	No final de cada fase do ciclo de desenvolvimento

Em suma os obstáculos e resistências ao desenvolvimento de *software* utilizando metodologias ágeis estão a ser ultrapassados [Bastos, 2013], as organizações investem em cursos de formação para os seus colaboradores que demonstram interesse em dominar uma vasta gama de tecnologias e metodologias, incluindo as ágeis que tanto se encontram em voga.

O mercado cada vez mais pede por agilidade, os resultados estão à vista e as empresas pretendem reduzir os prazos e custos de produção aumentando a qualidade dos seus produtos [Bastos, 2013].

2.1.4 Metodologia de Desenvolvimento de *Software* da *LabOrders*

A *LabOrders* não tem uma metodologia de desenvolvimento de *software* bem definida. Os programadores implementam uma nova funcionalidade e após a sua conclusão criam uma publicação no *Redmine* (*software open-source* [código fonte disponibilizado publicamente e que pode ser modificado] que serve para gerir projetos baseados na *web* e erros*)*. A publicação em questão apenas dá conta da necessidade de testar aquela funcionalidade e muitas vezes não contém qualquer descrição ou então esta é demasiado vaga e ambígua.

O analista de garantia da qualidade (em Inglês *quality assurance analyst*), um dos papéis que o autor da tese desempenhou na empresa, necessita invariavelmente de falar com os programadores para saber ao certo o que é suposto testar e muitas vezes nem os próprios programadores se lembram exatamente do que é preciso fazer porque implementaram a funcionalidade em questão há muito tempo. Desta forma acumulam-se imensos testes (quando o projeto de mestrado teve início havia largas dezenas de testes funcionais pendentes que foram codificados como trabalho suplementar, uma vez que no contexto da tese de mestrado apenas era solicitado codificar testes demonstrativos das novas funcionalidades que foram implementadas). Para além disso o código não é escrito a pensar nos testes. Relativamente aos testes funcionais, a maioria dos elementos *web* criados não possuem qualquer identificador, sendo muitas vezes necessário recorrer a seletores CSS (*Cascading Style Sheets*) para encontrar os elementos em questão, como se irá verificar posteriormente no capítulo **"Descrição Técnica"**. Isto traz um grande inconveniente, se os atributos CSS do elemento forem modificados é necessário alterar todos os testes que utilizam esse elemento. Relativamente aos testes unitários, grande parte das funções/métodos criados lançam exceções em vez de terem um valor de retorno o que inviabiliza o processo de teste dessas mesmas funções/métodos, uma vez que para realizar uma asserção é necessário um valor de retorno. Entende-se por asserção uma proposição que é sempre verdadeira. Como tal foi sugerida a XP principalmente devido ao TDD, uma prática de Engenharia específica a adotar por parte da *LabOrders*, o que foi aceite. Os testes são codificados antes das funcionalidades, ou seja, é o código da funcionalidade que se deve adaptar ao teste e não o oposto. Desta forma resolve-se o problema da ausência de identificadores únicos nos elementos *web* e de valores de retorno em funções/métodos porque a pessoa que desenvolve a funcionalidade será a mesma que a testará, não apenas no final mas sim durante todo o ciclo de desenvolvimento.

Em suma, a adoção da XP pela *LabOrders* irá reduzir o tempo gasto em desenvolvimento e testes, aumentando a produtividade da empresa.

2.1.5 Programação Extrema

A Programação Extrema (XP) é uma metodologia ágil para equipas de pequena e média dimensão que desenvolvem *software* baseado em requisitos vagos e que se modificam rapidamente [Beck, 1999], isto é, em situações onde o cliente nem sempre tem a certeza daquilo que pretende e ao longo do tempo pode alterar ou acrescentar requisitos. Distingue-se das outras metodologias principalmente porque requer *feedback* constante não apenas do código (através dos testes) como também do cliente, encorajando a comunicação pessoal.

O primeiro projeto a utilizar XP foi o *Chrysler Comprehensive Compensation System* (C3) que pretendia substituir várias aplicações de folha de pagamentos por um único sistema [Highsmith et al., 2000]. Teve início em 1993 utilizando metodologias de desenvolvimento de *software* tradicionais e em 1996 ainda não havia sido impresso nenhum comprovativo de salário através do novo sistema. Kent Beck foi contratado nesse ano como consultor para melhorias de *performance* tendo sido brevemente nomeado líder técnico. O projeto ficou pronto em pouco mais de um ano utilizando XP [Highsmith et al., 2000].

Os principais objetivos da XP são a satisfação do cliente e o cumprimento dos prazos. Para isso possui quatro valores fundamentais [Beck, 1999]:

- **Comunicação**: a XP promove um bom relacionamento entre clientes e programadores através de reuniões presenciais em detrimento de contacto por correio eletrónico, telefone e outras vias de comunicação.
- **Simplicidade**: o código deve ser simples, com o menor número de classes/métodos possível, procurando implementar apenas as funcionalidades essenciais do projeto. Como a XP suporta desenvolvimento evolucionário, os requisitos são sempre mutáveis. Sendo assim é preferível efetuar modificações posteriormente do que implementar um sistema muito complexo, com funcionalidades que podem nunca vir a ser utilizadas.
- ***Feedback***: é obtido através dos testes efetuados durante todo o ciclo de desenvolvimento. O cliente deve ter uma versão funcional do *software* para que possa sugerir novas funcionalidades e reportar erros que serão corrigidos nas próximas versões. Desta forma garante-se que o produto final estará de acordo com as necessidades e os requisitos definidos pelo cliente.
- **Coragem**: é preciso coragem para implementar a XP. Nem todas as pessoas têm facilidade em se exprimir oralmente e os problemas de relacionamento são inevitáveis. Para além disso todo o trabalho desenvolvido está constantemente sob escrutínio do cliente.

A XP baseia-se nas 12 boas práticas seguintes [Beck, 1999]:

- **Planeamento:** compreende o levantamento de requisitos funcionais e não funcionais (apenas as funcionalidades essenciais devem ser consideradas), tudo o que puder ser adiado (funcionalidades secundárias) são requisitos futuros e devem ser descartados nesta fase. Inclui também a definição de prazos e da arquitetura do sistema, bem como a distribuição de tarefas entre as diversas equipas/programadores. Em suma define quem faz o quê, como e quando.
- **Entregas frequentes:** as versões de avaliação para o cliente devem ser entregues mensalmente ou no máximo a cada 2 meses, com o menor tamanho possível e as funcionalidades devem ser implementadas por ordem decrescente de importância. Isto simplifica o *software*, possibilita o *feedback* rápido do cliente e maximiza a probabilidade do produto final estar de acordo com as suas necessidades e requisitos.
- **Metáfora:** é a descrição do *software* sem jargão técnico de modo a ser compreensível pelo cliente durante o processo de desenvolvimento.
- **Projeto simples:** a XP visa desenvolver *software* simples, com o menor número de classes/métodos possível e satisfazer apenas os requisitos atuais (funcionalidades essenciais) tendo em vista o cumprimento dos prazos. Os requisitos futuros (funcionalidades secundárias) devem ser adicionados à medida que forem surgindo.
- **Testes:** os testes devem ser criados antes do desenvolvimento das funcionalidades.
  Programação em pares: a implementação das funcionalidades é feita por grupos de duas pessoas, cada qual com o seu posto de trabalho. Um elemento controla o computador e escreve o código enquanto o outro tenta detetar erros de sintaxe e lógica, sempre no sentido de tornar o *software* mais eficaz, eficiente e intuitivo. Os programadores devem alternar os seus papéis ao longo do tempo.
  A principal vantagem da programação em pares é a possibilidade dos programadores aprenderem um com o outro.
- **Refatorização:** consiste na reestruturação do código sem alterar o seu comportamento externo. Entende-se por comportamento externo o que o *software* faz e por comportamento interno como o faz. A refatorização deve ser feita ao longo de todo o ciclo de desenvolvimento sempre que um dos elementos do par chegue à conclusão que é possível simplificar o código sem afetar nenhuma funcionalidade.
- **Propriedade coletiva:** todos os autores são responsáveis de igual forma pelo *software*. Como tal qualquer elemento da equipa pode fazer alterações a qualquer classe/método mesmo que não tenha sido quem inicialmente o desenvolveu, desde que codifique todos os testes associados necessários. Desta forma todos os programadores estão comprometidos com o projeto e na eventualidade de alguém abandonar a equipa antes da sua conclusão, é possível continuar o desenvolvimento sem grandes dificuldades.
- **Integração contínua:** consiste em integrar o código dos diferentes programadores e compilar o *software* várias vezes por dia. Desta forma toda a equipa se encontra em sintonia e a deteção de erros nos testes torna-se mais fácil uma vez que apenas são integradas pequenas modificações de cada vez. Caso um teste falhe a correção deve ser feita pela última pessoa a integrar o seu código. É recomendado que apenas exista um computador para realizar a integração do código, a que todos os elementos da

equipa devem ter acesso. Caso contrário podem surgir conflitos de versões (por exemplo, vários programadores a integrarem código em simultâneo).

- **40 Horas de trabalho semanal:** a XP defende que não se deve fazer horas extra por mais de duas semanas consecutivas. Caso isso aconteça existe um problema sério no projeto que deve ser resolvido através de um melhor planeamento (redistribuição de tarefas, redefinição de prazos de entrega, entre outros) ou da contratação de mais pessoal. Segundo o Manifesto Ágil os indivíduos e interações têm prioridade sobre os processos e ferramentas, tal como as respostas rápidas às mudanças têm prioridade sobre o planeamento. Sendo assim nestas situações deve-se sempre alterar os planos em vez de sobrecarregar os trabalhadores.
- **Cliente presente:** o cliente deve ter disponibilidade para comparecer às reuniões com os programadores, esclarecer as dúvidas existentes e avaliar o *software* ao longo de todo o ciclo de desenvolvimento. Desta forma evitam-se atrasos no projeto e interpretações erradas dos requisitos que podem levar à implementação de funcionalidades que não correspondem totalmente àquilo que é pretendido. Em alguns projetos o cliente chega mesmo a ser parte integrante da equipa de desenvolvimento.
- **Código padrão:** definição de regras de escrita de código e comentários, indentação, nomes de variáveis e constantes, *software* autorizado, entre outras, para que o trabalho possa ser compartilhado por todos os programadores.

Para assegurar o *feedback* constante, a XP possui uma prática específica de programação, o Desenvolvimento Guiado por Testes (TDD), que irá ser abordado em detalhe no capítulo seguinte.

2.1.5.1 Desenvolvimento Guiado por Testes

O Desenvolvimento Guiado por Testes (TDD) refere-se a "um estilo de programação em que três atividades estão fortemente interligadas: codificação, testes (na forma de testes unitários) e *design* (na forma de refatorização) " [Agile Alliance b].

"The act of writing a unit test is more an act of design than verification"
(Robert C. Martin)

"Walking on water and developing software from a specification are both easy if both are frozen"
(Edward V. Berard)

É uma abordagem evolucionária (iterativa e incremental) ao desenvolvimento de *software* que "combina *Test-First Development* (TFD), onde os testes são escritos antes do desenvolvimento ser realizado, e refatorização" [Beck, 2002].

Pode ser descrito sucintamente pelo seguinte conjunto de regras:

- Escrever um teste para uma unidade (menor parte testável) de uma funcionalidade (normalmente um método).
- Correr o teste que deve falhar porque o *software* ainda não possui essa funcionalidade.

- Escrever apenas código suficiente, o mais simples possível, para fazer com que o teste passe.
- Refatorizar o código e prosseguir com o desenvolvimento.
- Repetir o processo acumulando testes unitários ao longo do tempo [Agile Alliance b].

O diagrama da **Figura 2** fornece uma visão dos passos do TFD [Karai, 2009].

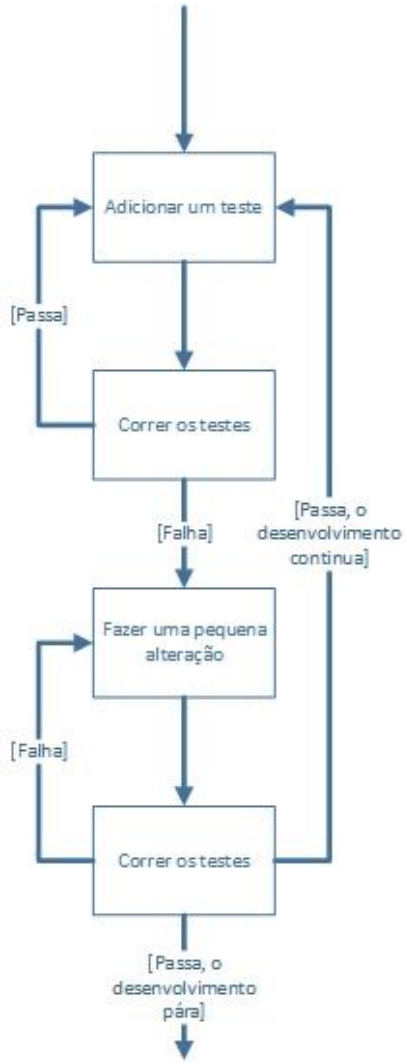

Figura 2- Os passos do TFD
TDD = TFD + Refatorização [Karai, 2009]

2.1.5.2 Refatorização

A refatorização é parte integrante do TDD e consiste num processo simples em que se fazem modificações estruturais ao código em passos pequenos, independentes e seguros (que não colocam em risco as funcionalidades do *software* como um todo). O código modificado é testado após cada um desses passos para assegurar que o seu comportamento não foi alterado, ou seja, que ainda funciona da mesma forma apesar de estar estruturado de maneira diferente. Destina-se a remover a redundância e ajuda a garantir que o *design* e a lógica são muito claros. São exemplos de técnicas de refatorização a procura de padrões e reestruturação do código bem como o agrupamento de funções/métodos com parametrização e objetivos semelhantes.

É importante compreender que apesar da refatorização partilhar alguns atributos e ser guiada por boas práticas da depuração e/ou otimização, na verdade não são a mesma coisa. Entende-se por depuração a técnica que permite identificar e reduzir erros de código. Por sua vez a otimização consiste em tornar o *software* o mais eficaz e eficiente possível, maximizando o seu desempenho e minimizando os recursos necessários para tal, através de fórmulas e modelos matemáticos, por exemplo.

Listam-se de seguida as principais diferenças entre refatorização e depuração/otimização:

- A refatorização não se foca apenas em corrigir erros de código.
- Reforçar o código de tratamento de erros não é refatorização.
- A adição de qualquer código defensivo não é considerada refatorização. Entende-se por código defensivo aquele que faz o *software* comportar-se de uma forma previsível mesmo sob circunstâncias imprevistas (dados de entrada ou ações do utilizador não esperadas, por exemplo).
- Aprimorar o código para o tornar mais testável (removendo dependências desnecessárias, funções/métodos redundantes, entre outros) também não é refatorização.

O processo de refatorização consiste geralmente numa série de atividades distintas que são abordadas de seguida:

- Em primeiro lugar identificar onde o *software* deve ser refatorizado, ou seja, descobrir quais os excertos de código que podem aumentar o risco de falhas ou erros.
- De seguida, determinar que refatorização deve ser aplicada aos excertos de código identificados.
- Garantir que a refatorização aplicada preserva o comportamento do *software*. Este é o passo crucial em que, dependendo do tipo de *software*: de tempo real, embebido ou de segurança crítica, por exemplo, medidas têm de ser tomadas para preservar o seu comportamento antes de o submeter à refatorização.
- Aplicar a técnica de refatorização apropriada.

- Avaliar o efeito da refatorização na qualidade das características do *software*, por exemplo a complexidade, a compreensibilidade, a manutenção, entre outras. Fazer o mesmo para o processo, por exemplo, a produtividade, os custos e esforços, entre outros.
- Garantir que a consistência dos requisitos é mantida entre o código do programa refatorizado e a documentação [Maruvada, 2014].

Os pontos seguintes resumem os passos importantes que devem ser respeitados quando se refatoriza uma aplicação:

1. **Em primeiro lugar formular os casos de testes unitários para a aplicação** que devem ser desenvolvidos de forma a testar o comportamento da aplicação e garantir que este permanece intacto após cada ciclo de refatorização.
2. **Identificar a abordagem à tarefa para refatorização** – isto inclui dois passos essenciais:
   - Encontrar o problema – averiguar se existem situações no excerto de código atual que possam aumentar o risco de falhas ou erros, e em caso afirmativo, em que consiste o problema.
   - Avaliar/decompor o problema – depois de identificar o problema potencial, avaliá-lo tendo em conta os riscos envolvidos.
3. **Desenhar uma solução adequada** – pensar sobre qual será o resultado após submeter o código à refatorização e formular uma solução de acordo com esse resultado, que será útil no processo de transição do código no estado atual (problemático) para o estado resultante pretendido (sem erros).
4. **Alterar o código** – proceder com a refatorização do código sem modificar o seu comportamento externo.
5. **Testar o código refatorizado** – para garantir que os resultados e/ou o comportamento são consistentes. Se o teste falhar reverter as alterações feitas e repetir a refatorização de uma forma diferente.
6. **Continuar o ciclo com os passos acima mencionados** até o código no estado atual (problemático) passar para o estado pretendido (sem erros).

Quando a técnica de refatorização é aplicada de uma forma disciplinada traz os seguintes benefícios, para além do que já foi alcançado com o TFD [Maruvada, 2014]:

- Aumenta a extensibilidade global do *software*.
- Reduz e otimiza os custos de manutenção do código.
- Facilita a escrita de código altamente padronizado e organizado.
- Garante que a arquitetura e o *design* do sistema são melhorados internamente mantendo o comportamento.
- Garante três fatores essenciais: legibilidade, compreensibilidade e modularidade do código.
- Garante uma melhoria constante à qualidade global do sistema.

2.1.5.3 Benefícios Esperados e Obstáculos/Dificuldades

O TDD possui muitas vantagens e também algumas desvantagens. Do ponto de vista do desenvolvimento, enumeram-se de seguida alguns benefícios chave que podem ser alcançados sem problemas por ter TDD implementado de uma forma disciplinada:

- Escrever testes requer acima de tudo que se considere verdadeiramente o que se quer do código. Isto ajuda principalmente na obtenção de um entendimento completo dos requisitos a serem simulados em linha com o propósito implícito do código.
- Promove a criação de uma especificação detalhada uma vez que os testes unitários resultantes são simples e funcionam como uma boa documentação para o código.
- Envolve um ciclo de *feedback* curto que reduz o tempo gasto em correções/modificações.
- Permite que o *design* evolua e se adapte à dinâmica de mudança do problema/requisitos.
- Promove uma simplificação radical do código, por exemplo, apenas é escrito código em resposta aos requisitos dos testes [Maruvada, 2014].
- Promove o desenvolvimento de código de alta qualidade.
- Fornece provas concretas de que o código funciona.
- Garante que o *design* é simples, limpo e intuitivo, concentrando-se na criação de operações que podem ser invocadas e que são testáveis.
- Suporta desenvolvimento evolucionário [Karai, 2009].
- Muitas equipas relatam reduções significativas nas taxas de defeitos à custa de um aumento moderado do esforço de desenvolvimento inicial.
- As mesmas equipas tendem a relatar que esse sacrifício inicial é mais do que compensado por uma redução do esforço em fases finais de projetos.
- Apesar da pesquisa empírica ainda não o ter conseguido confirmar, trabalhadores experientes relatam que o TDD leva a uma melhor qualidade do *design* do código e, de uma forma mais geral, leva a um maior grau de qualidade técnica, por exemplo melhorando as métricas de coesão e acoplamento [Agile Alliance b].

Em termos de desvantagens, um dos principais obstáculos é a relutância que alguns programadores mostram inicialmente em adotar esta técnica, afirmando que têm de escrever mais código do que é normal e que a implementação de testes é uma perda de tempo.

A curva de aprendizagem é um pouco longa, o que também pode contribuir para esse argumento [Sousa, 2013].

"If it's worth building, it's worth testing.
If it's not worth testing, why are you wasting your time working on it?"
(Scott Ambler)

Para linguagens de programação sem uma *framework* de testes unitários, a implementação do TDD pode tornar-se uma tarefa difícil uma vez que será preciso criar a dita *framework* de raiz. Entende-se por *framework* um conjunto de ferramentas e a forma como estas se inter-relacionam para conseguirem realizar uma determinada operação.

Também é árduo de implementar quando o *software* foi desenvolvido utilizando linguagens de programação que já não são mais suportadas ou quando não existe uma documentação de suporte para aplicações mais antigas. Sem exemplos práticos nem documentação não é possível tirar o máximo proveito da tecnologia ou ter um conhecimento aprofundado sobre todas as funcionalidades das aplicações.

O programador que vai realizar as alterações normalmente não sabe o que o código faz, receando naturalmente que as modificações a um excerto de código num determinado local possam ter repercussões noutro local [Sousa, 2013].

Os erros individuais típicos na implementação do TDD incluem:

- Esquecer-se de executar os testes com frequência.
- Escrever muitos testes de uma só vez.
- Escrever testes que são muito extensos ou que têm muitas dependências.
- Escrever testes demasiado triviais, por exemplo omitindo asserções.
- Escrever testes para código trivial, por exemplo acessores de instância.

Os erros típicos das equipas de programadores na implementação do TDD incluem:

- Adoção parcial – apenas alguns programadores da equipa utilizam o TDD.
- A falta de manutenção da bateria de testes – normalmente resultando numa bateria de testes com um tempo de execução demasiado longo e proibitivo.
- Bateria de testes abandonada (ou seja, raramente ou nunca executada) – por vezes como resultado de má manutenção, outras vezes como resultado da rejeição da equipa [Agile Alliance b].

Os benefícios da utilização do TDD superam claramente os seus obstáculos/dificuldades. O facto de ser necessário escrever mais código do que em abordagens tradicionais não é motivo suficiente para desprezar as vantagens de obter um produto final com menos erros e menores períodos de manutenção.

Certamente que o TDD não é a solução para todos os problemas mas ajuda os programadores a melhorarem as suas capacidades e a entregarem soluções mais confiáveis, modulares, flexíveis e fáceis de manter que correspondem às necessidades do cliente [Sousa, 2013].

2.2 Metodologia do Ciclo de Vida dos Testes Automáticos

A Metodologia do Ciclo de Vida dos Testes Automáticos (*Automated Test Lifecycle Metodology*, ou ATLM) (**Figura 3**) é uma abordagem sistemática para maximizar a cobertura dos testes [Haria].

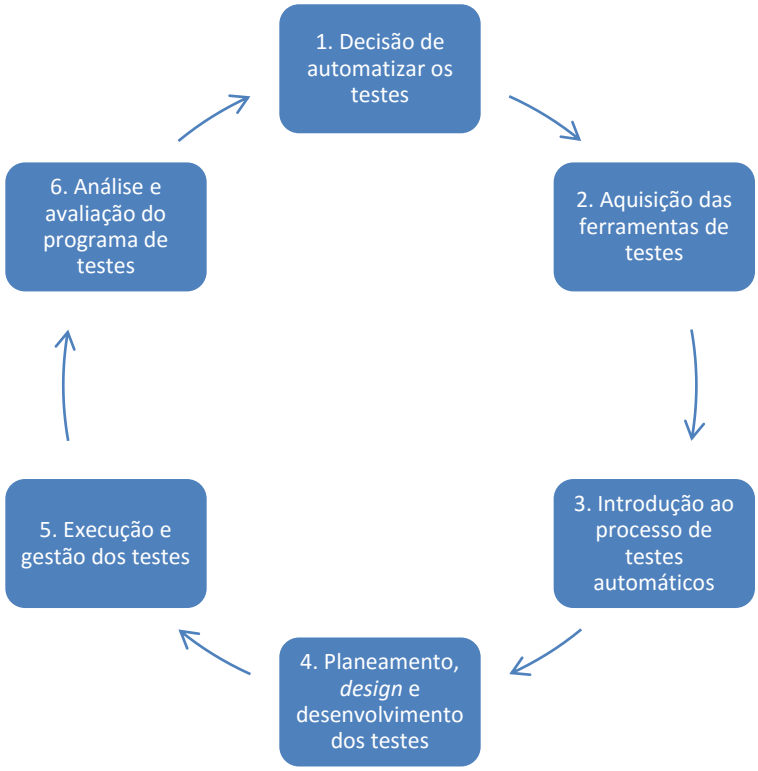

Figura 3 - Metodologia do Ciclo de Vida dos Testes Automáticos [Haria]

O primeiro passo no ATLM é tomar a decisão de automatizar os testes. Nesta fase a equipa realça os benefícios potenciais da automação e também cria uma proposta para a ferramenta de testes.

O segundo passo é a aquisição das ferramentas de testes. São selecionadas as ferramentas que se adequam ao ambiente de desenvolvimento e é produzida uma lista de critérios de avaliação. A ferramenta é escolhida com base nesses critérios e o vendedor é então contactado para a fornecer.

O terceiro passo é a introdução ao processo de testes automáticos, ou seja, é o procedimento necessário para introduzir os testes automáticos com sucesso num novo projeto. É constituído por duas partes. A primeira parte é a análise do processo de testes. Isto garante que a estratégia global de testes está em ordem e adaptada ao projeto. A segunda e última parte é a consideração da ferramenta de testes a utilizar onde se verifica quais as ferramentas de testes que poderão trazer alguma vantagem para o projeto em questão.

O quarto passo é o planeamento, *design* e desenvolvimento dos testes. Esta é a fase mais importante pois inclui a identificação das normas dos procedimentos de teste e a definição dos testes e do padrão de desenvolvimento. O plano de testes contém os resultados de todas as outras fases do ATLM. Para além disso é identificado o *hardware*, *software* e rede necessários para suportar o ambiente de testes juntamente com uma agenda de testes preliminar. A etapa do *design* dos testes inclui o número de testes a realizar, a forma de abordagem aos testes e as condições de teste que serão utilizadas. Finalmente a etapa de desenvolvimento é realizada de forma a que os testes automáticos possam ser reutilizáveis, reproduzíveis e sustentáveis. A definição de padrões auxilia imenso a criação de testes na medida em que prevê todos os cenários possíveis. Cada metodologia tem os seus próprios padrões. Um exemplo de padrão é o *Institute of Electrical and Electronic Engineers* (IEEE) 829, também conhecido como padrão 829 para documentação de testes de *software*.

O quinto passo é a execução e gestão dos testes. Todos os planos de testes são executados nesta fase de acordo com as normas discutidas previamente no planeamento.

O sexto passo é a análise e avaliação do programa de testes. Esta etapa é realizada ao longo do ciclo de vida de forma a existir uma melhoria contínua e menos falhas no final. Isto reduz os custos uma vez que é mais barato resolver falhas encontradas nas fases iniciais do que em fases posteriores.

Em suma o ATLM "é uma metodologia estruturada para assegurar a implementação bem-sucedida de testes automáticos" [Haria] [Dustin, 2001].

2.3 Razões para Automatizar o Teste de *Software*

Testar manualmente é uma tarefa comum e trabalhosa que pode levar a erros humanos uma vez que pequenos (mas importantes) detalhes do *software* podem ser facilmente esquecidos. Os testes automáticos validam que o *software* desenvolvido corresponde aos requisitos e ao que a organização pretende, assegurando desta forma a sua qualidade.

A automação também ajuda a evitar os defeitos e falhas. Por exemplo, se o ambiente de desenvolvimento está configurado incorretamente ou está a ser instalada a versão errada do *software* isso irá causar um erro, o que significa que o sistema irá funcionar mal ou falhar.

Outra razão para automatizar é que existem vários parâmetros não funcionais que precisam de ser testados, por exemplo: *performance*, usabilidade, segurança, escalabilidade, entre outros. Estes parâmetros são subjetivos e variam de pessoa para pessoa de acordo com o seu nível de tolerância. Por exemplo, a *performance* de um sistema pode ser aceitável para um indivíduo com uma tolerância elevada e inaceitável para outro indivíduo com um nível de tolerância mais reduzido. A automação é capaz de fornecer a vasta gama de tolerância necessária [Haria] [Sousa, 2009] uma vez que é possível codificar vários testes com diferentes níveis de tolerância (baixo, médio e elevado, por exemplo) de forma a satisfazer todos os perfis dos clientes.

2.4 Vantagens e Desvantagens dos Testes Automáticos de *Software*

É importante escolher as ferramentas certas para todos os aspetos de uma empresa. No ramo da Informática em particular, a opção pelos testes automáticos em detrimento dos manuais traz muitas vantagens. Os testes automáticos realizam as mesmas operações de cada vez que são executados, reduzindo assim o erro humano. Cada teste também pode ser repetido em diferentes versões de uma aplicação, poupando desta forma tempo e dinheiro. Isto resulta em mais testes executados em menos tempo, utilizando menos recursos e dando uma boa cobertura de testes. Significa também que a automação é mais rápida e acaba por ser mais barata do que os testes manuais [Haria].

Embora os testes automáticos tenham muitas vantagens também têm as suas desvantagens. Uma grande desvantagem é o facto de inicialmente ser necessário um investimento elevado para comprar as ferramentas e dar formação ao pessoal para as utilizar [Haria]. Para além disso, nas etapas de preparação dos testes, é preciso mão-de-obra elevada. Outro aspeto é que muitas áreas não são abrangidas pelos testes. Ainda não existem ferramentas suficientes para abranger todos os testes e por isso alguns têm de ser feitos manualmente.

Finalmente são necessárias capacidades de programação elevadas para escrever os *scripts* de automação dos testes, por isso é caro desenvolver as ferramentas para os tornar personalizáveis [Haria] [Galin, 2003]. Entende-se por *script* um programa que é executado através de outro programa.

2.5 Teste de Caixa-Preta vs Teste de Caixa-Branca

Teste de caixa-preta (em Inglês *black-box testing*) é um método de teste de *software* em que a estrutura/*design*/implementação internos do item a ser testado não são conhecidos pelo programador que está principalmente preocupado com a validação do *output* e não como este é produzido (a funcionalidade do item sob teste não é importante do ponto de vista do programador), ou seja, testa o comportamento externo do *software*. Entende-se por *output* os dados gerados por um computador.

Teste de caixa-branca (em Inglês w*hite-box testing*) é um método de teste de s*oftware* em que a estrutura/*design*/implementação internos do item sob consideração são conhecidos pelo programador que os valida juntamente com o *output*, ou seja, testa o comportamento interno do *software*.

São necessários conhecimentos de programação e implementação (estrutura interna e funcionamento) no teste de caixa-branca mas não são precisos no teste de caixa-preta.

O teste de caixa-preta é efetuado pela equipa profissional de testes e pode ser feito sem conhecimento da codificação interna do item. O teste de caixa-branca é geralmente efetuado pelos programadores que desenvolveram o item ou por programadores que têm uma compreensão do funcionamento interno do item [Patton, 2005] [Williams, 2006].

2.6 Testes Funcionais e Testes Unitários

Os testes funcionais destinam-se a examinar a funcionalidade global do *software*. A especificação para estes testes é muito detalhada já que todos os aspetos do sistema estão a ser testados incluindo as *interfaces*, logo envolve os clientes e utiliza testes alfa. Os testes alfa são testes de um novo pacote de *software* que são realizados pelos clientes no *site* da empresa ou do programador responsável pelo desenvolvimento [Haria]. Os testes funcionais enquadram-se na categoria de testes de caixa-preta.

Os testes unitários são fiáveis e rápidos de desenvolver uma vez que apenas testam pequenos componentes individuais (unidades) de uma aplicação, normalmente uma única classe ou, menos frequentemente, um pequeno grupo de classes fortemente relacionadas, formando um único conceito lógico [Millikin, 2014] [Test Driven Websites, 2010].

O atributo característico dos testes unitários é o isolamento completo, isto é, todas as dependências externas devem ser isoladas. A necessidade deste forte isolamento deve-se a dois objetivos chave: **O primeiro objetivo é uma aplicação composta por componentes livres de erros.** Para isto é necessária uma cobertura de testes o mais elevada possível, o que implica que se tenha muitos testes que devem primeiro ser codificados e de seguida executados repetidamente. Como tal os testes têm de ser rápidos e fáceis de codificar. O evitar da instanciação de uma vasta árvore de classes interligadas em cada teste e de conexões demoradas a recursos externos (bases de dados, serviços *web*, sistema de ficheiros) é exatamente o que torna os testes rápidos. O isolamento de recursos externos simplifica igualmente a configuração dos testes (não há necessidade de carregar e limpar ficheiros) o que aumenta ainda mais a velocidade dos testes. O isolamento total também torna a codificação dos testes mais fácil. A complexidade de um teste (combinações possíveis de parâmetros de entrada, condições de fronteira [conjuntos de valores máximos e mínimos em que os testes passam ou falham]) depende exponencialmente do número de classes inter-relacionadas que participam nesse teste. Limitando todos os testes a uma única classe, sem dependências, pode diminuir o número de casos de teste necessários para testar exaustivamente um sistema por uma ordem de magnitude. Isto é, o número de testes para uma aplicação de grandes dimensões irá ser menor se existir isolamento das dependências a recursos externos. Para além disso também reduz significativamente o esforço necessário para preparar o ambiente de testes.

O segundo objetivo é uma aplicação fácil de modificar. A aplicação evolui ao longo do tempo. Os requisitos e o *design* mudam. O código é refatorizado. Isto assume particular importância quando se adota a XP ou outras metodologias ágeis similares com refatorização constante e agressiva. Para tornar esta evolução contínua possível, o código e os testes não podem ter fraca qualidade nem ser difíceis de modificar. A chave é alta coesão e baixo acoplamento. Uma alteração na aplicação apenas deve exigir a modificação do código de uma única classe (ou de um grupo muito restrito de classes). Isto permite modificar a aplicação com um esforço relativamente baixo. Para além disso, uma mudança numa classe não deve fazer com que os

testes de nenhuma outra classe deixem de funcionar – isto permite modificar a aplicação de uma forma controlada (e também reduz o esforço).

O isolamento total dos testes unitários é um grande passo para alcançar este *design* modular: se for possível testar exaustivamente uma classe sem interagir com nenhuma das suas dependências, normalmente também é possível modificar essa classe sem ter de alterar testes de nenhuma das outras classes, impedindo que estes deixem de funcionar [Test Driven Websites, 2010].

Para além do isolamento, idealmente os testes unitários devem possuir as 4 qualidades seguintes:

- **Decisivo**: tem todas as informações para determinar o sucesso/insucesso.
- **Válido**: produz um resultado que corresponde à finalidade da unidade sob teste.
- **Completo**: contém toda a informação de que precisa para correr corretamente com um determinado sistema e unidade.
- **Repetitivo**: dá sempre os mesmos resultados se o sistema e a unidade forem os mesmos, ou seja, é determinístico.

Estes testes compreendem a maior porção dos testes automáticos de uma aplicação porque podem ser facilmente incorporados no processo de integração contínua [Millikin, 2014]. Entende-se por integração contínua uma prática de desenvolvimento de *software* em que os programadores integram o seu código frequentemente, existindo sempre múltiplas integrações diárias ou, no mínimo, uma integração por dia.

Deve-se ter tantos testes unitários quanto possível e testar a totalidade das *interfaces* públicas de todas as classes, excetuando talvez métodos assessores como *Get* e *Set* sem nenhuma lógica adicional [Test Driven Websites, 2010].

Dependendo de como os testes unitários são escritos tanto se podem enquadrar na categoria de testes de caixa-branca como na categoria de testes de caixa-preta. Se o programador estiver a testar dentro da aplicação, é um teste de caixa-branca. Se estiver a testar como uma pessoa externa ao sistema efetuando as chamadas através da *Interface* de Programação de Aplicações (em Inglês *Application Program Interface*, ou API) pública, então é um teste de caixa-preta [Beck, 2007]. Entende-se por API o conjunto de rotinas, protocolos e ferramentas para construir aplicações de *software*.

2.7 Importância da Escolha das Ferramentas de Testes Automáticos

Há muitos aspetos que precisam de ser tidos em consideração para escolher a melhor ferramenta para a realização de testes automáticos:

- O primeiro é a **facilidade de integração**. A ferramenta deve integrar-se facilmente com o sistema e a relação custo/*performance* deve ser ponderada. Para além disso a ferramenta tem de ser compatível com o *design* e a implementação da aplicação, devendo ser capaz de identificar todos os objetos e as diferentes classes da mesma [Haria].
- O segundo aspeto que deve ser considerado é a ***performance***. A forma como a ferramenta se comporta no ambiente de desenvolvimento com o tráfego de rede e o *hardware* é muito importante. Caso existam incompatibilidades entre a ferramenta, o ambiente de desenvolvimento e o *hardware* podem afetar o desempenho do sistema e congestionar a rede, por exemplo.
- O terceiro aspeto é o **tipo dos testes** que influencia diretamente o tipo de ferramenta uma vez que existem ferramentas especializadas em determinados tipos de teste.
- O quarto aspeto é a **manutenção**. O suporte e as comunidades *online* também devem ser tidos em consideração. Se existir apoio suficiente os problemas serão resolvidos mais rapidamente.
- O quinto e último aspeto é que as ferramentas têm de ser **acessíveis** [Haria].

2.8 Estudo Comparativo de Ferramentas de Automação

Foi efetuado um estudo comparativo de ferramentas de automação para averiguar quais as ferramentas de testes funcionais e unitários mais indicadas para serem utilizadas no processo de desenvolvimento de *software* da *LabOrders* e justificar as respetivas tomadas de decisão.

2.8.1 Ferramentas de Testes Funcionais

Foram analisadas todas as 15 ferramentas (11 gratuitas e 4 pagas) que foram encontradas na ampla pesquisa realizada.

Ferramentas gratuitas:

- ***Selenium WebDriver* com *Hypertext Preprocessor* (PHP) *Bindings*** [http://www.seleniumhq.org/projects/webdriver/]
- ***WebDriverJS*** [https://code.google.com/p/selenium/wiki/WebDriverJs]
- ***Satix*** [http://www.binpress.com/app/satix-seleniumbased-automation-testing-in-xml/1958]
- ***PhantomJS*** [http://www.phantomjs.org/]

- ***Zombie.js*** [http://zombie.labnotes.org/]
- ***Watir WebDriver*** [http://watirwebdriver.com/]
- ***Protractor*** [http://angular.github.io/protractor/#/]
- ***CasperJS*** [http://casperjs.org/]
- ***DalekJS*** [http://dalekjs.com/]
- ***Goutte*** [https://github.com/FriendsOfPHP/Goutte]
- ***Mink*** [https://github.com/Behat/Mink]

Ferramentas pagas:

- ***Sahi*** [http://sahipro.com/]
- ***Unified Functional Testing*** **(UFT)** [http://www8.hp.com/pt/pt/software-solutions/unified-functional-automated-testing/]
- ***TestingWhiz*** [http://www.testing-whiz.com/]
- ***Telerik Test Studio*** [http://www.telerik.com/teststudio]

De seguida serão apresentadas em maior detalhe cada uma delas. Relativamente às ferramentas gratuitas:

2.8.1.1 Selenium WebDriver com PHP *Bindings*

A *framework* utilizada pela *LabOrders* quando o trabalho de mestrado teve início era o *Selenium WebDriver* com PHP *bindings*. Entende-se por *bindings* a API que permite utilizar uma biblioteca numa linguagem de programação específica (neste caso PHP em vez de *Java*[1], linguagem original da API *WebDriver*). Por sua vez biblioteca (de *software*) é a implementação das rotinas e protocolos (regras) de uma API.

O *Selenium WebDriver* é um controlador de *browser*, ou seja, permite controlar o *browser*, simular as interações do utilizador e retornar informação das páginas do *browser*. Mostra o que está a acontecer no ecrã. Apenas possibilita efetuar testes funcionais (a empresa utiliza o *PHPUnit* [*framework* de testes unitários em PHP] juntamente com o *Selenium WebDriver* para fazer asserções e eventuais testes unitários sem dependências). É independente de Ambiente de Desenvolvimento Integrado (em Inglês *Integrated Development Environment* ou IDE) apesar de ser recomendado o uso do *Selenium* IDE ou outro. O IDE que estava a ser utilizado na *LabOrders* aquando do início deste projeto era o *NetBeans*, que continua, de resto, a ser utilizado.

[1] O nome *Java* não é um acrónimo, a equipa responsável decidiu batizá-la desta forma devido à pessoa que cunhou o seu nome estar a beber uma chávena da café de *Java* quando teve a ideia] [Murphy, 1996].

A *framework* existente disponibilizava um conjunto de funcionalidades não nativas já implementadas, nomeadamente a espera por pedidos AJAX (*Asynchronous JavaScript And* XML [*eXtensible Markup Language*]) e as mensagens de erro personalizadas no *findElementBy()*, um método que permite encontrar e obter elementos *web* através dos seletores do *Selenium WebDriver*.

Em suma, o *Selenium WebDriver* é considerado o futuro da automação *web open-source* e está prestes a tornar-se um padrão *World Wide Web Consortium* (W3C), a principal organização de padronização da *World Wide Web* (WWW). Tem a desvantagem de necessitar do *Selenium Server* para correr testes em aplicações remotas.

2.8.1.2 WebDriverJS

A *LabOrders* tinha interesse em obter alguns elementos *web* através de código *jQuery* (biblioteca *JavaScript* gratuita e *open-source* que permite navegar em documentos HTML [*HyperText Markup Language*], manipular eventos, criar animações e adicionar interações AJAX a páginas web) porque permite realizar pesquisa invertida (por exemplo obter o formulário que contém uma determinada *checkbox*), como tal foi considerado o *WebDriverJS* que é o *port JavaScript* da API *WebDriver* (*Java*) e é mais fiável do que os *JavaScript bindings*.

No entanto essa não era uma necessidade real visto que é possível obter o formulário diretamente sem passar pela *checkbox* utilizando os seletores do *WebDriver*. Tal como o *Selenium WebDriver* apenas possibilita efetuar testes funcionais (apesar de ser possível realizar testes unitários utilizando o *JsUnit* [*framework* de testes unitários em *JavaScript*], por exemplo).

2.8.1.3 Satix

O *Satix* é um *headless browser emulator*, ou seja, é uma implementação pura de especificações *HyperText Transfer Protocol* (HTTP) que envia pedidos HTTP reais para uma aplicação e analisa o conteúdo da resposta. Entende-se por HTTP o protocolo utilizado pela *web* que define como as mensagens são formatadas e transmitidas, e que ações os servidores *web* e os *browsers* devem tomar em resposta a vários comandos. Os testes são executados num *browser* invisível (apesar de ser possível tirar capturas de ecrã) e os resultados são apresentados na linha de comandos e/ou armazenados em relatórios. Tem a vantagem de ser mais rápido do que os controladores de *browser* e a desvantagem de não mostrar tudo o que está a acontecer no ecrã. É uma *framework Java* e os testes são codificados em XML. Apenas possibilita efetuar testes funcionais. Utiliza a API *WebDriver* e não depende de nenhum IDE contudo a sua licença gratuita apenas permite testar uma aplicação. Existem duas licenças pagas: a *Multi App License* para 5 aplicações que custa 499.99$ e a *Developer License* para aplicações ilimitadas que custa 999.99$.

2.8.1.4 PhantomJS

O *PhantomJS* é um *headless Webkit* programável com uma API *JavaScript* utilizado para correr os testes através de um executor de testes apropriado. Entende-se por *Webkit* o motor de renderização HTML/CSS dos *browsers Chrome* e *Safari*, os outros *browsers* não suportam o *Webkit* nativamente já que têm os seus próprios motores de renderização (*Internet Explorer* – *Trident*, *Firefox* – *Gecko*, *Opera* – *Presto*, entre outros). Por sua vez executor de testes é a ferramenta utilizada para correr os testes. Cada *framework* tem o seu executor de testes (o da maior parte das *frameworks* baseadas em *WebDriver* é o *GhostDriver*, por exemplo).

Não necessita do *Selenium Server* e torna a *framework* independente de IDE apesar de ter de ser utilizado em conjunção com o *CasperJS*, o *DalekJS* ou o *Sahi*, por exemplo, que vão ser analisados posteriormente. Para além dos testes funcionais também possibilita efetuar testes unitários.

2.8.1.5 Zombie.js

O *Zombie.js* é um *headless browser emulator* cujos testes são escritos em *JavaScript*. É independente de IDE e suporta *promises* (o eventual resultado de uma operação assíncrona). No entanto tem muitas dependências: necessita do *Node.js*, *Node Package Manager* (NPM), um compilador *C++* e *Python*. Entende-se por NPM o Gestor de Pacotes do *Node.js*, a ferramenta de linha de comandos que permite interagir com o repositório *online* do *Node.js* e que é necessária para a instalação de pacotes, gestão de versões e dependências [nodebr, 2013]. Por sua vez compilador é um *software* que traduz o código fonte para código máquina de forma a poder ser interpretado pelo computador. Para além dos testes funcionais também possibilita efetuar testes unitários.

2.8.1.6 Watir WebDriver

O *Watir WebDriver* é a segunda *framework* de testes mais popular e mais utilizada a seguir ao *Selenium WebDriver*. Pode ser usado tanto como um controlador de *browser* ou como um *headless browser emulator*, é independente de IDE, a espera por pedidos AJAX é uma funcionalidade nativa e possui outras funcionalidades como a espera por animações *JavaScript*. Os testes são codificados em *Ruby*[2]. Para além dos testes funcionais também possibilita efetuar testes unitários.

[2] Tal como acontece com o *Java*, o nome *Ruby* não é um acrónimo tendo sido sugerido ao criador da linguagem numa sessão de conversação *online*, e foi escolhido por o rubi ser a pedra preciosa do mês de nascimento de um dos seus colegas [Maeda, 2002].

2.8.1.7 Protractor

O *Protractor* é uma *framework* de testes para aplicações *AngularJS* apesar de também poder ser utilizado em *sites* não angulares. Entende-se por *AngularJS* a *framework open-source JavaScript* mantida pela *Google* para simplificar o desenvolvimento e teste de aplicações *web* de página única: https://angularjs.org.

O *Protractor* é um controlador de *browser* e utiliza a API *WebDriver*. É independente de IDE e a espera automática é uma funcionalidade nativa. Os testes são codificados em *JavaScript*. Para além dos testes funcionais também possibilita efetuar testes unitários. Tem a desvantagem de necessitar do *Selenium Server*.

2.8.1.8 CasperJS

O *CasperJS* é um *headless browser emulator* cujos testes são codificados em *JavaScript* ou *CoffeeScript* (versão simplificada do *JavaScript*, compila em *JavaScript*). É independente de IDE e é utilizado em conjunção com o *PhantomJS*. Para além dos testes funcionais também possibilita efetuar testes unitários.

2.8.1.9 DalekJS

O *DalekJS* é uma ferramenta *open-source* cujos testes são codificados em *JavaScript*. É o que mais se aproxima de uma solução *cross-browser*, suporta o *Chrome*, *Internet Explorer*, *Safari* e *Mobile Safari* (no futuro irá suportar *Desktop Opera*, *Chrome* no *Android* e o *browser* do *Blackberry*). Pode ser utilizado tanto como um controlador de *browser* ou como um *headless browser emulator*, é independente de IDE e é utilizado em conjunção com o *PhantomJS*. Não possui uma *interface web*, apenas *interface* da linha de comandos e ainda se encontra em fase de desenvolvimento (versão provisória de testes para programadores), não devendo ser utilizado para produção. Para além dos testes funcionais também possibilita efetuar testes unitários.

2.8.1.10 Goutte

O *Goutte* é um *headless browser emulator* cujos testes são codificados em PHP. É independente de IDE. Apenas possibilita efetuar testes funcionais.

2.8.1.11 Mink

O *Mink* foi desenvolvido pelos criadores do *Zombie.js* e funciona como uma ponte entre os diversos *drivers* (*GoutteDriver* [*Goutte*], *SeleniumDriver* [*SeleniumRC*], *Selenium 2 Driver* [*Selenium WebDriver*], *WebDriver* [*Satix*, *Watir WebDriver*, *Protractor*], *SahiDriver* [*Sahi*], *ZombieDriver* [*Zombie.js*], entre outros). Neste contexto entende-se por *driver* o *software* que permite comunicar com as *frameworks*.

O *Mink* tanto pode ser utilizando como um controlador de *browser* ou como um *headless browser emulator* e é independente de IDE. Para além dos testes funcionais também possibilita efetuar testes unitários.

Em relação às ferramentas pagas:

2.8.1.12 Sahi

Os criadores do *Sahi* afirmam que se trata de uma ferramenta que funciona em todos os *browsers* (mas não contemplam versões móveis). Os testes são codificados em *Sahi Script*, *Java* ou *Ruby*. A espera por pedidos AJAX é uma funcionalidade nativa. Resolve alguns problemas que o *Selenium* tem com o *Internet Explorer* (*IEFrames*, janelas *popup* [que aparecem subitamente no ecrã], entre outros). Permite correr 5 instâncias paralelas numa única máquina sem necessitar de múltiplas máquinas virtuais. Entende-se por máquina virtual um sistema operativo ou aplicação que está instalada em *software* que imita *hardware* dedicado. O *Sahi* tanto pode ser utilizando como um controlador de *browser* ou como um *headless browser emulator* e é independente de IDE apesar de ser recomendada a utilização do *Sahi Textpad* ou do *Sahi edit mode* (*jEdit*). Para além dos testes funcionais também possibilita efetuar testes unitários. Disponibiliza uma versão de um mês para testes mas a versão paga custa 695$ por utilizador por ano.

2.8.1.13 Unified Functional Testing

O *Unified Functional Testing* (UFT), anteriormente conhecido como *QuickTest Professional* (QTP) é um *software* da *Hewlett-Packard* (HP) cujos testes são codificados em *VBScript*. Possui uma *Interface* Gráfica do Utilizador (em Inglês *Graphical User Interface* ou GUI) e componentes de testes reutilizáveis. Para além dos testes funcionais também possibilita efetuar testes unitários. Existem dois tipos de licenças: *Seat License*, para um computador, e *Concurrent License*, para múltiplos computadores. A HP disponibiliza gratuitamente uma licença de dois meses para testes mas o custo de uma *Seat License* oscila entre os 8000$ e os 12000$ nos Estados Unidos da América.

2.8.1.14 TestingWhiz

O *TestingWhiz* permite executar testes em múltiplos dispositivos a partir de um único servidor centralizado. Possui um escalonador de testes integrado e uma *interface* gráfica que elimina a necessidade de conhecimentos de programação. No entanto os testes podem ser codificados sendo que o *TestingWhiz* suporta mais de 120 linguagens de programação [Cygnet Infotech, 2013]. Para além dos testes funcionais também possibilita efetuar testes unitários. Pode ser solicitada uma versão de testes e existem várias licenças: mensais, anuais, perpétuas, entre outras, cujo preço varia de acordo com o número de licenças adquiridas e as necessidades do cliente. O custo das licenças anuais e perpétuas varia entre os 449$ e os 1649$.

2.8.1.15 Telerik Test Studio

O *Telerik Test Studio* é a ferramenta de testes mais intuitiva do mercado, segundo os seus criadores. Possui um *plugin* para o *Visual Studio* (programa que pode ser instalado e utilizado juntamente com o *Visual Studio*) e os testes são codificados em *C#/VB.net*. Para além dos testes funcionais também possibilita efetuar testes unitários. A licença que permite executar testes funcionais denomina-se *Test Studio Functional* e tem um custo de 169$ por mês. Existe também uma licença *Test Studio Load* para testes de carga (que permitem identificar os limites de capacidade de uma aplicação), que custa 79$ por mês e a licença *Test Studio Ultimate* que combina os benefícios das duas outras licenças e custa 199$ por mês.

2.8.2 Comparação e Escolha da Ferramenta de Testes Funcionais

Foram consideradas várias ferramentas para o processo de codificação e execução dos testes funcionais na *LabOrders*, conforme foi referido no subcapítulo anterior.

O *WebDriverJS* possui sobre o *Selenium WebDriver* com PHP *bindings* a vantagem dos testes serem codificados em *JavaScript*, possibilitando a obtenção direta de elementos *web* através do método *executeScript()*. No entanto chegou-se à conclusão que essa não era uma verdadeira necessidade já que o *Selenium WebDriver* permite obter esses mesmos elementos *web* de uma forma mais simples através dos seus seletores. Para além disso também é possível obter atributos dos elementos *web* diretamente a partir de código *JavaScript* e do método *executeScript()*. A obtenção dos elementos *web* em si é mais complicada uma vez que o método *executeScript()* retorna um objeto *stdClass* que necessita de ser convertido para o tipo *WebElement* de forma a poder ser manipulado, o que foi conseguido recorrendo à técnica de *Reflection* do PHP como iremos ver em detalhe no capítulo **"Descrição Técnica"**.

Os *headless browser emulators* foram tidos em consideração por serem mais rápidos a executar os testes do que os controladores de *browser*. No entanto a codificação e modificação dos testes tornar-se-ia mais complexa uma vez que não seria possível observar o que ocorre no ecrã. O tempo ganho na execução dos testes não justifica o tempo perdido na sua criação.

O *Watir WebDriver* e o *Protractor* têm o benefício da espera por pedidos AJAX ser uma funcionalidade nativa ao contrário do que acontece com *o Selenium WebDriver*, no entanto esta e outras funcionalidades foram implementadas posteriormente após a adoção inicial do *Selenium*.

O *DalekJS* permite correr os testes em versões móveis dos *browsers* o que seria útil para testar os *sites WordPress* (sistema de gestão de conteúdo *open-source* baseado em PHP e *MySQL*) com *templates* responsivos que a *LabOrders* desenvolve, por exemplo. Entende-se por *template* uma página *web* pré-desenvolvida que pode ser utilizada para qualquer pessoa introduzir o seu próprio texto e imagens. Contudo, o facto de ainda se encontrar em fase de testes para programadores não podendo ser utilizado para produção inviabilizou a sua escolha.

Todas as ferramentas pagas foram descartadas porque um dos principais critérios na escolha das ferramentas de testes automátiicos é o facto de serem acessíveis como vimos previamente no subcapítulo **"Importância da Escolha de Ferramentas de Testes Automáticos"**. Para além disso nenhuma destas ferramentas possui vantagens que justifiquem a sua opção em detrimento do *Selenium WebDriver*. Apesar do UFT e do *TestingWhiz* terem *interfaces* gráficas o *Selenium* também disponibiliza uma GUI através de IDEs como é o caso do *NetBeans* utilizado pela *LabOrders*.

Pode-se observar os critérios utilizados para a escolha da ferramenta de testes funcionais nas **Tabelas 2** e **3**. Foram considerados estes critérios porque a *LabOrders* pretendia um controlador de *browser* PHP gratuito com o mínimo de dependências e uma GUI. Preferencialmente também deveria suportar o maior número de *browsers* possível e permitir codificar tanto testes funcionais como unitários.

Tabela 2 – Comparação entre as ferramentas de testes funcionais (parte 1)

	Linguagem	**Tipo**	**Dependências**	**Tipo de testes**
Selenium WebDriver	PHP	Controlador de *browser*	*Selenium Server*, PHP *bindings*	Funcionais
WebDriverJS	*JavaScript*	Controlador de *browser*	*Selenium Server*	Funcionais
Satix	XML	*Headless browser emulator*	N. A.	Funcionais
PhantomJS	*JavaScript*, *Sahi Script*, *Java* ou *Ruby*	*Headless Webkit*	*CasperJS*, *DalekJS*, *Sahi*	Funcionais e unitários
Zombie.js	*JavaScript*	*Headless browser emulator*	*Node.js*, NPM, um compilador *C++* e *Python*	Funcionais e unitários
Watir WebDriver	*Ruby*	Controlador de *browser*/*Headless browser emulator*	N. A.	Funcionais e unitários
Protractor	*JavaScript*	Controlador de *browser*	*Selenium Server*	Funcionais e unitários
CasperJS	*JavaScript*	*Headless browser emulator*	*PhantomJS*	Funcionais e unitários
DalekJS	*JavaScript*	Controlador de *browser*/*Headless browser emulator*	*PhantomJS*	Funcionais e unitários
Goutte	PHP	*Headless browser emulator*	N. A.	Funcionais
Mink	Depende do driver	Controlador de *browser*/*Headless browser emulator*	Depende da *framework*	Funcionais e unitários
Sahi	*Sahi Script*, *Java* ou *Ruby*	Controlador de *browser*/*Headless browser emulator*	*PhantomJS* (para ser utilizado como *headless browser emulator*)	Funcionais e unitários
UFT	*VBScript*	Controlador de *browser*	N. A.	Funcionais e unitários
TestingWhiz	Suporta mais de 120 linguagens	Controlador de *browser*	N. A.	Funcionais e unitários
Telerik Test Studio	*C#* ou *VB.net*	Controlador de *browser*	N. A.	Funcionais e unitários

Tabela 3 – Comparação entre as ferramentas de testes funcionais (parte 2)

	Espera por pedidos AJAX	**GUI**	***Browsers***	**Preço**
Selenium WebDriver	Sim (funcionalidade não nativa já implementada na *framework* existente)	Não	Versões *Desktop*	Grátis
WebDriverJS	Não	Não	Versões *Desktop*	Grátis
Satix	Não	Não	Versões *Desktop*	Grátis para testar uma aplicação (499.99$ para 5 aplicações e 999.99$ para aplicações ilimitadas)
PhantomJS	N. A.	Não	Versões *Desktop*	Grátis
Zombie.js		Não	Versões *Desktop*	Grátis
Watir WebDriver	Sim (nativamente)	Não	Versões *Desktop*	Grátis
Protractor	Sim (nativamente)	Não	Versões *Desktop*	Grátis
CasperJS	Não	Não	Versões *Desktop*	Grátis
DalekJS	Não	Não	Versões *Desktop* e Móveis	Grátis
Goutte	Não	Não	Versões *Desktop*	Grátis
Mink	Não	Não	Versões *Desktop*	Grátis
Sahi	Sim	Não	Versões *Desktop*	695$ por utilizador por ano
UFT	Não	Sim	Versões *Desktop*	Entre 8000$ e 12000$ por computador
TestingWhiz	Não	Sim	Versões *Desktop*	Entre os 449$ (licenças anuais) e 1649$ (licenças perpétuas)
Telerik Test Studio	Não	Não	Versões *Desktop*	169$ por mês

Posto isto, a escolha final recaiu sobre o *Selenium WebDriver* com PHP *bindings* visto que é tecnologia mais utilizada e que a curto prazo irá ser o padrão da indústria, tendo consequentemente mais documentação e suporte disponíveis na *Internet*. A decisão de alterar a *framework* resultaria na necessidade de voltar a codificar todos os testes ou de manter várias versões.

2.8.3 Comparação e Escolha da *Framework* de Testes Unitários

Foram analisadas duas *frameworks* para a codificação e execução dos testes unitários na *LabOrders*:

- ***PHPUnit*** (https://phpunit.de/)
- ***SimpleTest*** (http://www.simpletest.org/)

Optou-se por analisar estas duas alternativas devido ao facto de serem as mais utilizadas para integração com o *Kohana*, a *framework* que a empresa usa para desenvolver a sua plataforma de encomendas *online*.

O *PHPUnit* é uma instância da arquitetura *xUnit* (*jUnit*, *JsUnit*, *cUnit*, entre outros) para *frameworks* de testes unitários e como tal não possibilita efetuar outros tipos de testes.

O *SimpleTest* tem a particularidade de permitir criar testes *web* para além dos testes unitários. Em ambos os casos os testes são escritos em PHP.

A escolha final recaiu sobre o *SimpleTest* em detrimento do *PHPUnit* (embora o projeto esteja preparado para utilizar ambos caso seja pretendido) por vários motivos, que irão ser apresentados de seguida. A *LabOrders* utiliza a versão 2.3.4 (desatualizada) do *Kohana*. A documentação já não se encontra disponível bem como o repositório do módulo dos testes unitários (*PHPUnit*). Também é impossível obter o instalador através da *Internet*, apenas se teve acesso ao mesmo a partir da pasta partilhada da empresa. Posto isto a utilização do módulo de testes unitários (*PHPUnit*) da versão 2.3.4 do *Kohana* não é viável visto que o repositório foi removido, não há qualquer documentação disponível e os exemplos incluídos são de testes unitários simples como a potência de um número, não existe qualquer exemplo com *mock objects*, necessários para o isolamento das dependências. Entende-se por *mock objects* objetos simulados que mimetizam o comportamento de objetos reais de uma forma controlada. Isto irá ser analisado em mais detalhe no capítulo **"Descrição Técnica"**. Ressalve-se que apesar do módulo de testes unitários ser baseado no *PHPUnit* a sintaxe é diferente.

A atualização do *Kohana 2* para *Kohana 3* está fora de questão por parte da *LabOrders* porque ocorreram muitas mudanças, a estrutura de diretórios é diferente o que implicaria a reescrita da maior parte da aplicação. Como tal a utilização do módulo de testes unitários (*PHPUnit*) do *Kohana 3*, para o qual já existe documentação, também não é uma opção.

Para além disso o *SimpleTest* tem:

- A instalação mais simples (extrair para o diretório e executar).
- O controlo de versões mais simples.
- Menos dependências e muitas extensões (*webtester* [testes web], *formtester* [formulários], *auth* [autenticação]).
- Um bom relatório de erros que é fácil de estender.
- Um resumo do relatório de erros.
- Um bom executor *web* AJAX, com execução de ficheiros únicos e grupos de ficheiros.
- Uma ferramenta de *diff* (que deteta e exibe as diferenças entre dois ficheiros) sem os problemas com espaços em branco e novas linhas da ferramenta de *diff* do *PHPUnit*.
- Um adaptador que permite correr testes do *SimpleTest* com o *PHPUnit* e vice-versa, ou seja, pode ser utilizado em conjunção com o *PHPUnit*.
- Pode correr através do *browser* ou da linha de comandos.
- Pode testar tanto estado (valores das variáveis) como comportamento e funcionalidades de *front-end* (com que os utilizadores interagem diretamente).
- Os resultados dos testes são personalizáveis.

O *SimpleTest* gera automaticamente o código das classes *mock* (para a criação de *mock objects*) ao contrário do *PHPUnit*, simplificando imenso o processo e poupando tempo ao programador e à empresa. Por exemplo, para a classe:

```
class DatabaseConnection {
    function DatabaseConnection() { }
    function query($sql) { }
    function selectQuery($sql) { }
}
```

Código 1 – Classe *DatabaseConnection*

É gerado o seguinte código:

```
class MockDatabaseConnection extends DatabaseConnection {
    public $mock;
    protected    $mocked_methods    =    array('databaseconnection',    'query',
selectquery');

    function MockDatabaseConnection() {
        $this->mock = new SimpleMock();
        $this->mock->disableExpectationNameChecks();
    }

    …
    function DatabaseConnection() {
        $args = func_get_args();
        $result = &$this->mock->invoke("DatabaseConnection", $args);
        return $result;
    }
```

```
    function query($sql) {
        $args = func_get_args();
        $result = &$this->mock->invoke("query", $args);
        return $result;
    }

    function selectQuery($sql) {
        $args = func_get_args();
        $result = &$this->mock->invoke("selectQuery", $args);
        return $result;
    }
}
```

Código 2 – Classe *MockDatabaseConnection* gerada pelo *SimpleTest*

Neste contexto (necessidade de trabalhar forçosamente com uma versão desatualizada do *Kohana*, de 2009), a única desvantagem do *SimpleTest* é não ser o padrão da indústria apesar de existir muita informação e exemplos, tanto no *site* oficial como no repositório, ao contrário do que acontece com o módulo de testes unitários (*PHPUnit*) do *Kohana 2*, para o qual não existe qualquer documentação disponível.

A **Tabela 4** fornece um resumo das vantagens e desvantagens do *SimpleTest* sobre o *PHPUnit* no contexto deste projeto.

Tabela 4 – *SimpleTest* vs *PHPUnit*

	SimpleTest	***PHPUnit***
Documentação	Disponível	Não disponível
Repositório	Existente	Não existente
Instalador	Disponível *online*	Não disponível *online*
Instalação	Mais simples	Mais complexa
Controlo de versões	Mais simples	Mais complexo
Dependências	Mais reduzidas	Mais numerosas
Ferramenta de *diff*	Sem problemas	Com problemas de espaços em branco e novas linhas
Adaptador para executar testes de outras *frameworks*	Sim	Não
Execução	Através do *browser*, linha de comandos e IDE	Através da linha de comandos e IDE
Testes	Unitários e funcionais	Apenas unitários
Geração automática das classes *mock*	Sim	Não
Padrão da indústria	Não	Sim

O *SimpleTest* irá ser utilizado para testes com *mock objects* sem dependência de base de dados e em conjunção com o *Phactory* (que irá ser abordado no próximo capítulo) para testes que requiram *mock database* (simulação da base de dados real). O *PHPUnit* também se encontra configurado e pode ser usado caso pretendido, no entanto, o *SimpleTest* e o

Phactory são muito mais simples, fáceis e rápidos de utilizar para testes com *mock objects* e *mock database*.

2.8.4 Comparação e Escolha da Abordagem aos Testes Unitários

Existem duas alternativas para abordar os testes unitários dependentes de dados externos:

- ***Database factory*** (*Phactory* [https://github.com/chriskite/phactory])
- ***Fixtures***

Uma *database factory* é um objeto que permite criar outros objetos a partir de registos de uma base de dados. Permite também modificar e eliminar esses registos. Por sua vez uma *fixture*, também denominada por contexto do teste, é o ambiente fixo e bem conhecido sob o qual os testes correm de forma a garantir a sua repetitividade.

Carregar uma base de dados com um conjunto específico de registos ou preparar dados de entrada e configurar/criar *mock objects* são exemplos de *fixtures.*

A opção da *database factory* foi considerada a mais viável devido a possuir diversas vantagens sobre as *fixtures*, para além de funcionar tanto com o *SimpleTest* como com o *PHPUnit*. A opção das *fixtures* apenas funcionaria com o *PHPUnit* já que o *SimpleTest* nativamente não permite carregar *fixtures*.

A *database factory* escolhida foi o *Phactory* devido a ser a única solução encontrada que suporta tanto o *SimpleTest* como o *PHPUnit*

Enumeram-se de seguida as vantagens da *database factory* (*Phactory*) sobre as *fixtures*:

- Os testes unitários são mais legíveis e fáceis de escrever.
- É possível modificar os objetos que são criados programaticamente em vez de ficar preso aos dados das *fixtures*, que são carregados todos ao mesmo tempo. Qualquer alteração à base de dados implica modificar o conteúdo das *fixtures*.
- O *Phactory* fornece um *Object-Relational Mapping* (ORM) simples e leve que pode ser utilizado nos testes em vez de obter os dados todos de uma vez através de uma biblioteca pesada como *Doctrine*. Entende-se por ORM a técnica de programação que permite conectar os objetos a uma base de dados relacional (é possível definir associações entre objetos). Também pode ser integrado com o ORM do *Kohana* [Kite, 2011].

Relativamente a desvantagens das *fixtures*:

- A manutenção é muito difícil, é muito confuso e frustrante controlar todos os diferentes cenários de que os testes necessitam.

- A falta de validação da lógica dos modelos pode fazer com que os testes deixem de funcionar.
- Não é possível criar objetos dinamicamente.
- O conceito de associações não existe.
- Tipicamente num formato pouco amigável como o XML.
- Também não existe qualquer tipo de suporte para obter e manipular os dados no teste. É assumido que todas as tabelas, gatilhos (*triggers*), sequências e vistas são criados antes de um teste ser executado. Entende-se por gatilho um procedimento que é executado quando ocorrem ações específicas numa base de dados. Por sua vez sequência é uma funcionalidade que permite criar valores únicos, por exemplo incrementar o valor da chave primária de uma tabela. Por fim vista é uma forma de exibir a informação de uma base de dados, é um subconjunto da base de dados obtido a partir de uma consulta. Apesar da definição da vista ser permanente, os dados que contém são dinâmicos dependendo do instante em que é acedida.
  Como não existe conceito de associações é necessário utilizar bibliotecas como *Doctrine 2* ou *Propel* para criar a estrutura da base de dados antes dos testes serem executados.

A **Tabela 5** resume as principais vantagens da *database factory* (*Phactory*) sobre as *fixtures*.

Tabela 5 – *Database factory* (*Phactory*) vs *Fixtures*

	***Database factory* (*Phactory*)**	***Fixtures***
Associações	Sim	Não
Testes	Mais legíveis e fáceis de escrever	Menos legíveis e difíceis de escrever
Criação dinâmica de objetos	Sim	Não
Manutenção	Fácil	Difícil, confusa e frustrante
Validação da lógica dos modelos	Sim	Não
Formato	Amigável	Pouco amigável
Suporte para obtenção e manutenção dos dados nos testes	Sim	Não
Obtenção dos dados	Objetos criados dinamicamente	Dados carregados todos de uma vez

Por todos estes motivos a decisão recaiu sobre a *database factory* (*Phactory*) em detrimento das *fixtures*.

3 Descrição Técnica

Neste capítulo são descritos, em detalhe, os aspetos técnicos de todo o trabalho desenvolvido.

O capítulo começa por apresentar as funcionalidades inovadoras a serem desenvolvidas para o cumprimento dos objetivos propostos pela *LabOrders*. De seguida são apresentadas a arquitetura e estrutura do primeiro projeto desenvolvido, a *framework* de testes funcionais e unitários sem dependências externas, mais particularmente os componentes que a constituem, as suas respetivas funções e a forma como se inter-relacionam. Seguidamente é detalhada a implementação das novas funcionalidades. É também descrita a correção da funcionalidade de envio do relatório de testes por correio eletrónico, que já havia sido criada pela empresa antes do projeto de mestrado ter tido início, e o desenvolvimento dos testes funcionais.

O segundo projeto a ser descrito é a *framework* de testes unitários com dependências externas (base de dados e serviços), e tal como no primeiro são abordadas a sua arquitetura e estrutura. É também detalhado o processo de criação de um novo módulo de testes unitários e o desenvolvimento de uma aplicação de testes como prova de conceito.

Foi ainda possível ir além dos objetivos propostos inicialmente com o desenvolvimento de trabalho suplementar e como tal é feita uma referência final aos Sistemas de Gestão de Conteúdos (*Content Management System* – CMS) criados, bem como à importação de catálogos de produtos para a plataforma *online* de encomendas da *LabOrders*.

3.1 Funcionalidades Desenvolvidas

A proposta inicialmente apenas contemplava o desenvolvimento de um único projeto que era uma *framework* de testes funcionais e unitários sem dependências externas. No entanto, após uma reunião em conjunto com o Supervisor Externo da empresa, Doutor Tiago Carvalho, e o Orientador do ISEP, Doutor Constantino Martins, foi também proposto o desenvolvimento de um segundo projeto, uma *framework* de testes funcionais com dependências externas (base de dados e serviços), que foi prontamente aceite tendo como propósito acrescentar uma mais-valia para a empresa e para a dissertação. Sendo assim, foram criadas duas *frameworks* separadas em vez de apenas uma por solicitação da empresa.

A *framework* de testes funcionais e unitários sem dependências externas testa a aplicação *web* (plataforma de encomendas *online* da *LabOrders*) através dos *browsers* e como tal não precisa de ter acesso ao código da plataforma (que se encontra num projeto à parte). No entanto, a *framework* de testes unitários com dependências externas já precisa de ter acesso ao código e à base de dados da plataforma.

A *LabOrders* pretende ter as duas *frameworks* separadas devido ao facto da *framework* de testes funcionais e unitários sem dependências externas ser para uso dos estagiários curriculares ao passo que a *framework* de testes unitários com dependências externas é para uso dos funcionários da empresa. Como tal não há necessidade de dar acesso ao código e à base de dados da plataforma a quem vai trabalhar com a primeira *framework*. No entanto são facilmente integráveis uma vez que a *framework* de testes funcionais não possui quaisquer dependências externas. Sendo assim e como ambas as *frameworks* foram configuradas no âmbito da tese de mestrado, para as integrar numa só basta importar os ficheiros do projeto da *framework* de testes funcionais para o projeto da *framework* de testes unitários com dependências externas (base de dados e serviços).

As funcionalidades inovadoras planeadas para cumprir os objetivos propostos pela *LabOrders* para o primeiro projeto, a *framework* de testes funcionais e unitários sem dependências externas, são as que a seguir se apresentam:

- ***Download* automático de ficheiros através do *Internet Explorer 9* (criação de *script AutoIT*):** o *Selenium WebDriver* não permite efetuar o *download* automático de ficheiros através do *Internet Explorer 9* porque não consegue manipular *IEFrames* (tipo de barra de notificação exclusivo das versões mais recentes do *Internet Explorer*). Apenas permite realizar o *download* automático de ficheiros através dos outros dois *browsers* utilizados pela *LabOrders*, o *Firefox* e o *Chrome*, pois estes não usam *IEFrames*. Como a empresa também tem a necessidade de executar os testes com o *Internet Explorer 9*, foi criada esta funcionalidade inovadora, que contempla a criação de um *script AutoIT* (linguagem *open source* para automatização de GUIs) para resolver o problema.
- **Exibição de mensagens de erro personalizadas**: quando o *Selenium WebDriver* não consegue obter um elemento retorna apenas uma mensagem de erro genérica *"Element not found"*. A *LabOrders* tinha falta de uma funcionalidade que permitisse

definir mensagens de erro mais explicativas e que identificassem imediatamente os elementos em questão, de forma a tornar a codifição dos testes mais rápida, simples e intuitiva.

- **Análise do conteúdo de ficheiros .pdf**: a empresa necessitava de analisar o conteúdo dos ficheiros .pdf gerados e descarregados durante a execução dos testes, para assegurar que estes eram criados com a informação correta. Por exemplo para um ficheiro .pdf referente a uma ordem de encomenda é preciso verificar se possui o nome do produto, o nome do fornecedor e o preço corretos.
- **Obtenção de elementos (e atributos) através de código *jQuery***: A *LabOrders* demonstrou interesse em obter alguns elementos *web* (e respetivos atributos) através de código *jQuery* devido à sua funcionalidade de pesquisa invertida (por exemplo, obter o formulário que contém um determinado botão). No entanto, no entender do autor da tese isso não é necessário pois é possível obter o formulário diretamente sem passar pelo botão, de uma forma mais simples e imediata, utilizando os seletores do *Selenium WebDriver*. De qualquer das formas a funcionalidade foi desenvolvida conforme solicitado pela empresa.

Foi também corrigida e melhorada a funcionalidade de envio do relatório de testes por correio eletrónico que tinha sido desenvolvida pela *LabOrders* antes do projeto de mestrado ter tido início. A mensagem com os resultados dos testes não estava a ser enviada juntamente com o ficheiro de anexo que contém as capturas de ecrã mas após as correções ficou a funcionar corretamente.

Relativamente ao segundo projeto, a *framework* de testes unitários com dependências externas (base de dados e serviços), foi realizado o trabalho que se apresenta de seguida:

- **Criação do novo módulo de testes unitários**: a *LabOrders* solicitou a criação deste módulo porque o *Kohana* apenas permite executar os ficheiros localizados no seu diretório *"\controllers"*. O módulo replica parcialmente a estrutura de ficheiros do *Kohana*, permitindo separar os controladores da aplicação e as baterias de testes. Para além disso também possibilita ter um subrepositório só para os testes, separado do repositório principal da aplicação.
- **Desenvolvimento da aplicação de testes unitários**: foi desenvolvida uma aplicação de testes unitários. A finalidade desta aplicação é fornecer à empresa exemplos de utilização de *mock objects* e *mock database*.

A implementação de todas estas funcionalidades irá ser detalhada em subcapítulos seguintes cujos nomes são os mesmos das funcionalidades a que dizem respeito.

Foi também realizada a configuração da *framework* de testes funcionais e unitários sem dependências externas, e da *framework* de testes unitários com dependências externas. Podem ser consultadas nos anexos **"Configuração do *Selenium WebDriver*, *PHPUnit*, *NetBeans* e WAMP"** e **"Configuração do *Kohana*, *SimpleTest* e *Phactory*"**, respetivamente. A

configuração é um passo essencial porque se as *frameworks* não forem configuradas não funcionam. A *LabOrders* já estava a utilizar o *Selenium WebDriver* antes do começo do projeto de mestrado, contudo, como havia sido configurado incorretamente apenas funcionava numa versão muito desatualizada do *Firefox* quando devia funcionar nas versões mais recentes dos três *browsers* que a empresa utiliza: *Firefox*, *Chrome* e *Internet Explorer*. Este problema foi resolvido com uma reconfiguração total e correta da *framework*.

Os *browsers* foram igualmente configurados de forma a permitirem o *download* automático de ficheiros através dos testes. A sua configuração pode ser consultada no anexo **"Configuração dos *Browsers*"**.

Em complemento dos 2 projetos principais foram ainda desenvolvidos projetos extra objetivos da tese mas no âmbito do estágio, nomeadamente a importação de catálogos de produtos e a criação de *sites* em *WordPress*, que podem ser consultados nos anexos **"Importação de Catálogos de Produtos"** e **"*Sites* em *WordPress*"**, respetivamente. Neste contexto foram realizadas tarefas como registo de utilizadores, integração de sistema de pagamentos *online*, animações de *scroll*, criação de conteúdo e *design* do *layout*, entre outras.

A metodologia de desenvolvimento seguida ao longo da tese foi logicamente a mesma que foi *aconselhada* à empresa no subcapítulo **"Metodologia de Desenvolvimento de *Software* da *LabOrders*"**, ou seja a XP. De acordo com o TDD, a prática de engenharia específica da XP referida previamente, os testes foram codificados antes das funcionalidades.

Os testes realizados para demonstração das novas funcionalidades implementadas foram:

- **Teste de *download* de um documento de concessão**: relativo à funcionalidade de *download* automático de ficheiros através do *Internet Explorer 9* (criação de *script AutoIt*). São criadas concessões na plataforma por projeto onde se define qual é o seu Investigador Principal (em Inglês *Principal Investigator* ou PI) e os privilégios dos utilizadores associados a esse mesmo projeto. Após a criação de uma concessão na plataforma é gerado um documento com a informação respetiva que é listado numa página própria para o efeito de forma a que possa ser descarregado. O teste navega até a essa página após realizar *login* como PI, efetua o *download* do primeiro documento e verifica se o ficheiro foi transferido corretamente.
- **Teste do conteúdo de uma ordem de encomenda**: relativo à funcionalidade de análise do conteúdo de ficheiros .pdf. O teste realiza o *login* como PI, navega até à página de encomendas, efetua o *download* do ficheiro .pdf referente à primeira ordem de encomenda e verifica se o nome do fornecedor, o nome do produto e o preço total da encomenda coincidem no ficheiro e na plataforma.

Para além destes testes foram ainda codificadas dezenas de outros testes como trabalho suplementar. Todos eles serão explicados e poderão ser visualizados no subcapítulo **"Desenvolvimento de Testes Funcionais"**. Como exemplo foi detalhado o funcionamento e

implementação do caso de teste dos serviços da plataforma de encomendas *online* da *LabOrders*.

Concluída a apresentação das funcionalidades desenvolvidas no âmbito da tese, o próximo capítulo começará por explicar a arquitetura do primeiro projeto desenvolvido, a *framework* de testes funcionários e unitários sem dependências externas.

3.2 Arquitetura da *Framework* de Testes Funcionais e Unitários Sem Dependências

A framework de testes funcionais e unitários é composta por três componentes: duas *frameworks* (*Selenium WebDriver*, *PHPUnit*) e uma biblioteca (*PDF2Text)* (**Figura 4**).

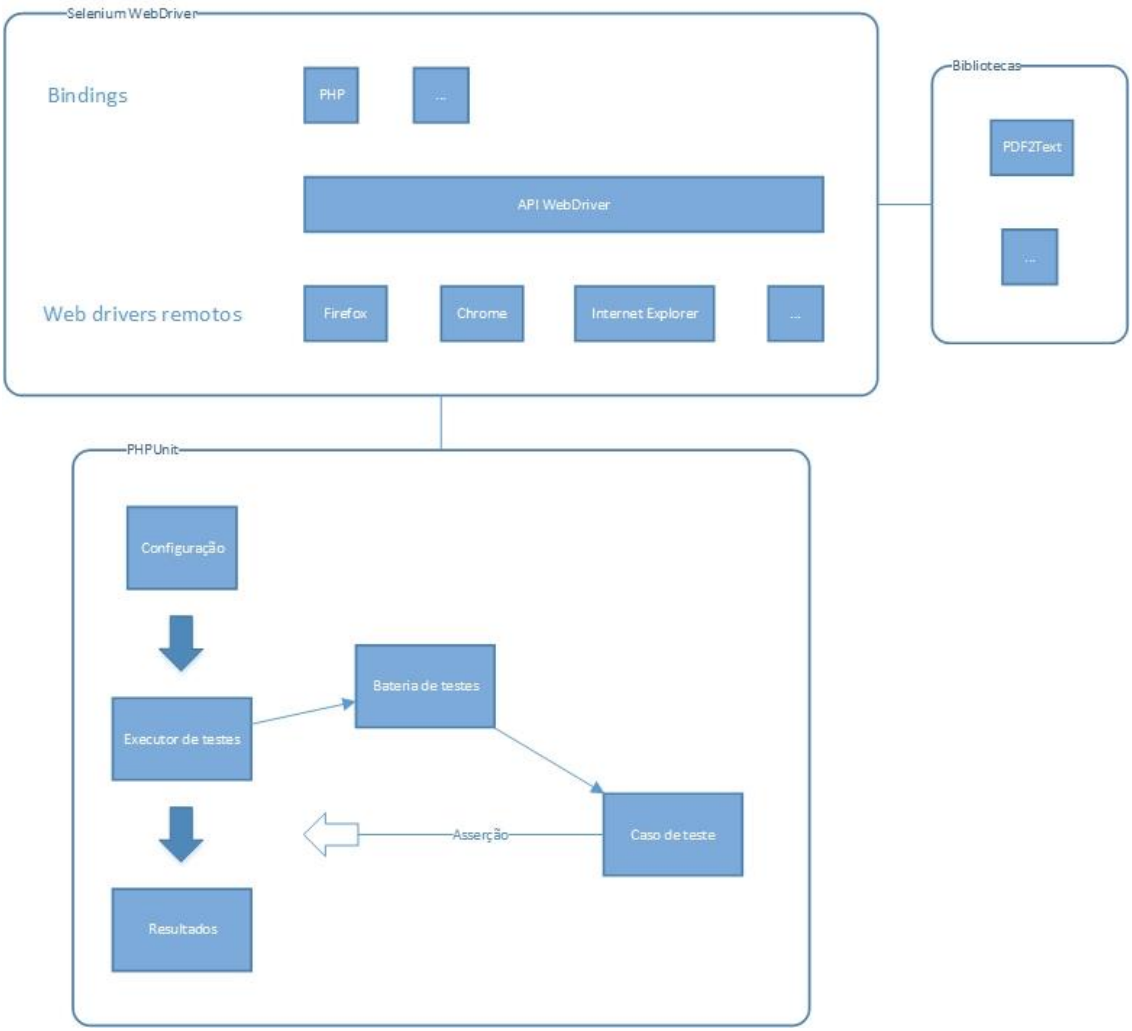

Figura 4 – Diagrama de arquitetura da *framework* de testes funcionais e unitários sem dependências

O *Selenium WebDriver* é utilizado para obter os elementos *web* e toda a informação das páginas necessária para a realização dos testes. A linguagem original da *API WebDriver* é o *Java*, logo na eventualidade de se querer codificar os testes noutra linguagem é preciso incluir os *bindings* respetivos. Como na *LabOrders* os testes são codificados em PHP foram incluídos os bindings PHP. O *Selenium WebDriver* possui um único *webdriver* remoto incluído de raiz, o *Firefox Driver*. Caso se queira executar os testes noutros *browsers* é preciso instalar e configurar os *drivers* respetivos. Como na *LabOrders* se pretende que os testes também corram no *Chrome* e no *Internet Explorer*, foram instalados e configurados o *Chrome Driver* e o *IEDriver*, respetivamente.

O *PHPUnit* é requerido pelo *Selenium WebDriver* para poder fazer as asserções utilizando a informação por si obtida. Possui um executor que após ser corretamente configurado permite correr as baterias de testes (grupos de casos de teste). Entende-se por caso de teste um conjunto de condições ou variáveis sob as quais o analista de garantia da qualidade irá determinar se uma aplicação se encontra a funcionar corretamente ou não. Tanto a configuração do *Selenium WebDriver* como a do *PHPUnit* podem ser consultadas no anexo **"Configuração do *Selenium WebDriver*, *PHPUnit*, *NetBeans* e WAMP"**.

A biblioteca *PDF2Text* foi necessária para o desenvolvimento da funcionalidade de análise do conteúdo de ficheiros .pdf. Permite obter o conteúdo de ficheiros .pdf em texto simples (sem formatação do .pdf) de forma a poderem ser realizadas asserções sobre esse mesmo conteúdo.

Apresentada a arquitetura da *framework* de testes funcionais e unitários sem dependências externas, os componentes que a constituem e respetivas funções, no capítulo seguinte irá ser descrita a estrutura desses mesmos componentes e a forma como se relacionam entre si.

3.3 Estrutura da *Framework* de Testes Funcionais e Unitários Sem Dependências

A estrutura da *framework* de testes funcionais e unitários sem dependências externas é representada através do diagrama da **Figura 5** que descreve a relação entre as classes dos componentes que a constituem. Devido à dimensão das classes que possuem largas dezenas de atributos e métodos não é possível incluí-los no diagrama por questões de espaço e visibilidade da imagem. Para além disso, cada teste possui os seus próprios atributos e métodos, a *framework* contém cerca de uma centena de testes logo é igualmente impossível colocá-los a todos no diagrama. Irão ser descritos pela classe ***TestClass***, que representa os casos de teste, e pela classe ***PageClass***, que representa as páginas que os testes manipulam.

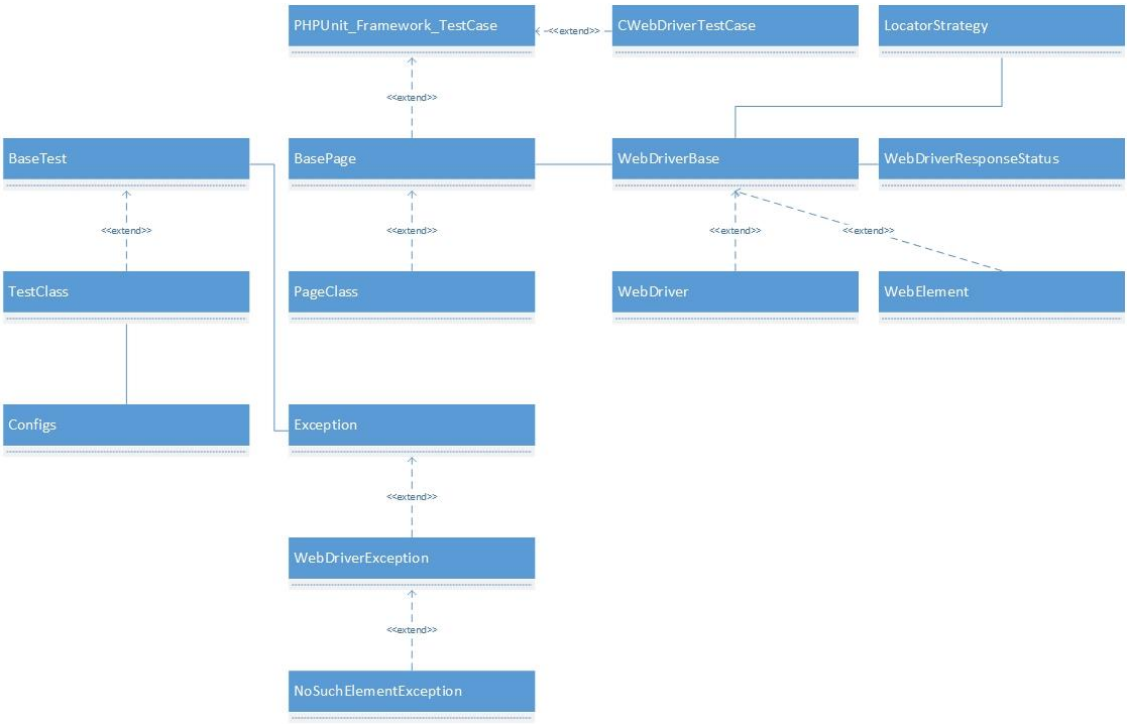

Figura 5 – Diagrama de classes da *framework* de testes funcionais e unitários sem dependências

Os testes são constituídos pelas classes ***BaseTest*** e ***BasePage***, juntamente com a ***TestClass*** e a ***PageClass*** referidas previamente. Todas as classes que compõem a *framework* irão ser descritas de seguida:

- ***Exception***: classe base para todas as exceções no PHP. Entende-se por exceção um evento que ocorre durante a execução de um programa e perturba o seu fluxo normal de instruções.
- ***PHPUnit_Framework_TestCase***: representa os casos de teste do *PHPUnit*.
- ***CWebDriverTestCase***: representa os casos de teste do *Selenium* (testes funcionais). Como o *Selenium WebDriver* precisa do *PHPUnit* para realizar as asserções, a classe ***CWebDriverTestCase*** é uma extensão de ***PHPUnit_Framework_TestCase***.
- ***WebDriverBase***: contém funções de que ambas as classes ***WebDriver*** e ***WebElement*** necessitam, nomeadamente a execução de pedidos ao servidor e manipulação das respetivas respostas, localização dos elementos *web*, entre outras. Esta classe foi modificada para o desenvolvimento da funcionalidade de exibição de mensagens de erro personalizadas conforme se poderá verificar posteriormente no capítulo com o mesmo nome.
- ***WebDriver***: engloba as funções da API *WebDriver* e permite estabecer a conexão ao *Selenium Server*, eliminar sessões, refrescar páginas, entre outras funcionalidades.
- ***WebElement***: corresponde aos elementos *web* e contém funções para a obtenção dos seus valores e atributos, obtenção e limpeza dos dados de caixas de texto, clique em botões, submissão de formulários, entre outras.
- ***LocatorStrategy***: constitui as estratégias de localização dos elementos *web*, por exemplo através do identificador único, nome da classe, texto do endereço, entre outras.
- ***WebDriverException***: representa as exceções do *WebDriver.*
- ***NoSuchElementException***: trata-se da exceção específica que ocorre quando não se consegue encontrar um elemento *web*. Esta classe, tal como a ***WebDriverException***, foi modificada para o desenvolvimento da funcionalidade de exibição de mensagens de erro personalizadas.
- ***Configs***: abrange as configurações do *Selenium WebDriver* como por exemplo o endereço do *site* que se pretende testar, o *browser* em que os testes vão correr, qual o servidor, as credenciais de *login* dos diferentes perfis de acesso, entre outras.
- ***BaseTest***: possui os atributos e métodos comuns à maioria dos testes, por exemplo a função que permite efetuar a parametrização inicial do teste ou a função que possibilita a eliminação da sessão e dos *cookies* no final do mesmo. Entende-se por *cookies* as mensagens transmitidas por um servidor *web* a um *browser* com o objetivo de identificar utilizadores e preparar páginas personalizadas para estes.
- ***BasePage***: possui os atributos e métodos comuns à maioria das páginas, por exemplo a função de *login* e os atributos referentes ao formulário de autenticação, a função que permite navegar para uma determinada página através do *menu* e os atributos referentes aos botões desse mesmo *menu*, entre outros.

- ***TestClass***: é uma das duas classes criadas de raíz para cada teste juntamente com a ***PageClass***. Contém as funções de teste que por sua vez invocam as funções da ***PageClass*** de forma a obter a informação das páginas que permite realizar as asserções.
- ***PageClass***: contém os atributos que representam os elementos *web* das páginas e as funções que permitem obter a informação necessária para se poder realizar os testes.

Apresentada a estrutura da *framework* de testes funcionais e unitários sem dependências externas, dos componentes que a constituem e respetivas funções, irá ser abordada no próximo capítulo a primeira funcionalidade desenvolvida, o *download* automático de ficheiros através do *Internet Explorer 9*.

3.4 *Download* Automático de Ficheiros através do *Internet Explorer 9*

O *Selenium WebDriver* não consegue identificar, obter e interagir com *IEFrames* (**Figura 6**), logo era impossível realizar o *download* automático de ficheiros através do *Internet Explorer 9* (e versões mais recentes) porque os testes não conseguem encontrar e clicar no botão Guardar Como (ou *Save As*) da *IEFrame* para guardar o ficheiro em questão.

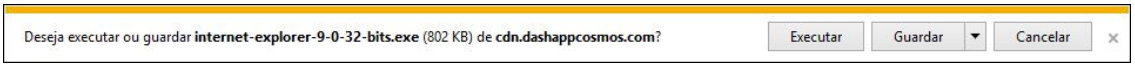

Figura 6 – Exemplo de *IEFrame* para *download* de um ficheiro

É possível desativar a *IEFrame* de *download* através da edição de registos:

1. Iniciar o Editor de registo (**Iniciar** -> **Executar** -> ***regedit.exe***).
2. Expandir *HKEY_CURRENT_USER*.
3. Navegar até *HKEY_CURRENT_USER**Software**Microsoft**Windows**Shell*.
4. Se a chave *AttachmentExecute* não existir deve ser criada (clique com o botão direito do rato, **Novo** -> **Chave** e inserir o nome *AttachmentExecute*).
5. Criar a chave {0002DF01-0000-0000-C000-000000000046} (clique com o botão do lado direito do rato, **Novo** -> **Chave** e inserir o nome {0002DF01-0000-0000-C000-000000000046}).
6. Criar um novo valor binário com o nome *AcroExch.Document* (para ficheiros .pdf) [botão do lado direito do rato -> **Novo** -> **Valor binário** e inserir o nome *AcroExch.Document*]. Para outros tipos de ficheiros o nome é diferente, por exemplo para ficheiros .xls é Excel.Sheet.8 e para ficheiros .xlsx é Excel.Sheet.12.
7. Reiniciar o *Internet Explorer*. A *IEFrame* será desativada.

No entanto desativando a *IEFrame* o comportamento por defeito passa a ser sempre abrir o ficheiro, não há forma de fazer com que a ação padrão seja o Guardar Como e por esse motivo não resolve o problema [Sturlese, 2013]. Foi então realizada uma pesquisa através da qual se chegou à conclusão que a criação ou adaptação de um *script* (*AutoIT*) seria a solução.

Foi encontrado o seguinte *script* para automatização do *download* de ficheiros através do *Internet Explorer 9*: https://gist.github.com/srinivasapulagam/3016355. O fluxograma da **Figura 7** representa o seu funcionamento.

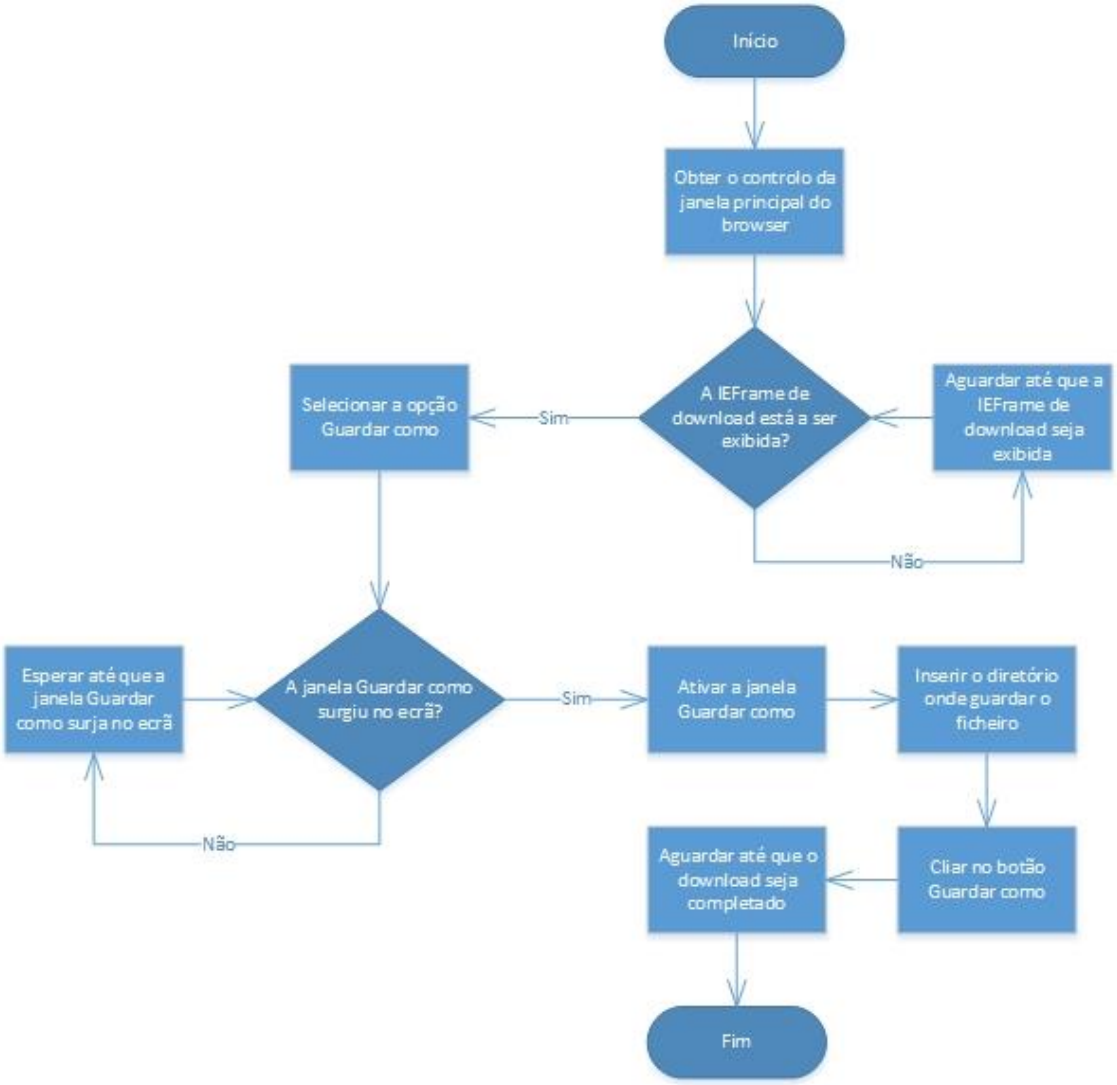

Figura 7 – Operação de *download* automático de um ficheiro no *Internet Explorer 9*

No entanto após a execução de testes de *download* de *ficheiros* concluiu-se que este *script* não é compatível com todas as versões do *Internet Explorer 9*, funcionando apenas em alguns contextos. Posto isto arranjou-se uma solução alternativa que consiste em utilizar um gestor de *downloads* (neste caso foi usado o *Orbit*) que não requira a interação do utilizador, ou seja, que detete automaticamente o início do processo de *download*. Desta forma deixa de ser necessário clicar no botão Guardar Como/*Save As* da *IEFrame* e passa a ser preciso clicar no botão *Download* do *Orbit*, que já é manipulável pelo *Selenium* por intermédio de um *script AutoIT* criado especialmente para o efeito, através dos seguintes passos:

1. Instalar o *Orbit*: http://www.orbitdownloader.com/pt/index.htm.

2. Instalar o *AutoIT*: https://www.autoitscript.com/site/autoit/downloads/.
3. Iniciar um *download* através do *Internet Explorer*. O *Orbit* deteta automaticamente o início do processo de *download* e exibe a janela da **Figura 8**:

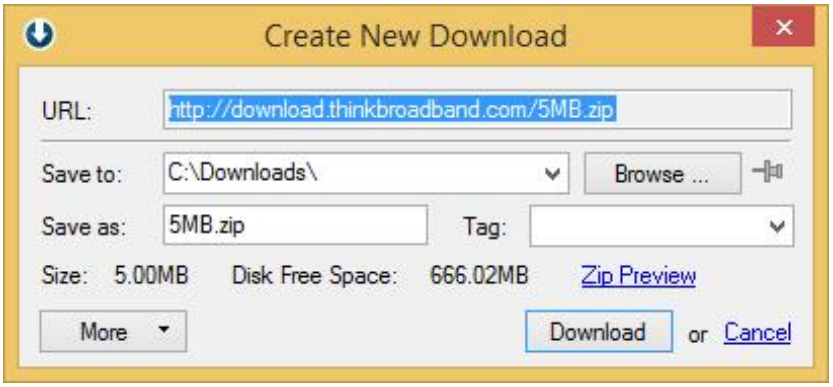

Figura 8 – Janela de *download* do *Orbit*

4. Identificar o botão ***Download*** exibido na figura acima:
    a. Abrir o ***AutoIT Window Info*** (**Figura 9**).

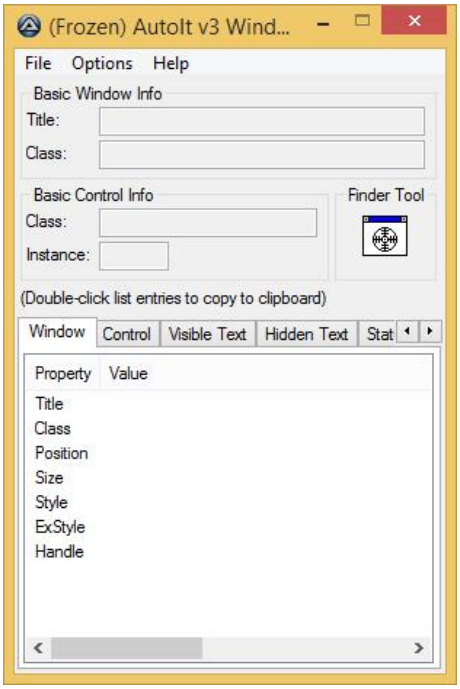

Figura 9 – *AutoIT Window Info*

b. Arrastar a ***Finder Tool*** para o botão ***Download*** de forma a preencher os dados do ***AutoIT Window Info*** da forma demonstrada na **Figura 10**.

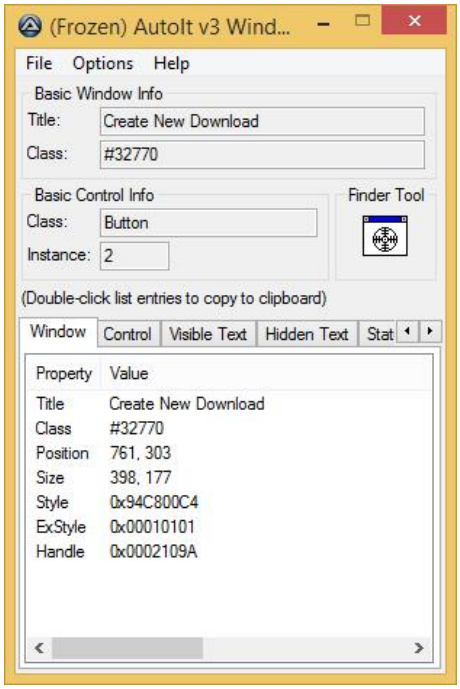

Figura 10 – *AutoIT Window Info* preenchido

5. Criar o *script AutoIT* com a informação da figura acima:
   a. Abrir o ***SciTE Script Editor*** do *AutoIT*.
   b. Foram utilizados 2 métodos, o `<WinWait("title", "text", timeout)>` para aguardar que a janela de *download* do *Orbit* surja após clicar na hiperligação do ficheiro, e o `<ControlClick("title", "text", "controlID")>` para clicar no botão ***Download*** do *Orbit*. O `"title"` tanto pode ser a '*Class*' como o '*Title*', o `"text"` é apenas descritivo (opcional), o `timeout` é o número (inteiro) de segundos que se pretende esperar e o `"controlID"` é a aglutinação do nome com a instância da classe. Isto resulta no seguinte *script*:

```
; Wait 10 seconds for the download window to appear
WinWait("[CLASS:#32770]", "", 10)

; Click on the Download button
ControlClick("[CLASS:#32770]", "", "Button2");
```

Código 3 – *Script AutoIt* para *download* automático de ficheiros através do *Internet Explorer 9*

6. Guardar o *script* no formato .au3: clicar em ***Save***, em **Nome de ficheiro:** introduzir o nome pretendido (neste caso foi *download*) e em **Guardar com o tipo:** selecionar ***AutoIt (au3)***.
7. Compilar o *script* .au3 e convertê-lo para um ficheiro executável (.exe): clicar com o botão direito do rato no ficheiro .au3 e selecionar ***Compile Script***. O ficheiro executável vai ser criado no mesmo diretório do ficheiro .au3.

Após a criação do *script* resta executá-lo nos testes que realizam *downloads* automáticos quando o *browser* definido no ficheiro de configuração do *Selenium* for o *Internet Explorer*. Esse ficheiro vem pré-configurado com o *Firefox* como *browser* padrão e apenas foi necessário modificá-lo no contexto deste capítulo. Segue-se um excerto do mesmo:

```
include_once 'bootstrap.php';

class Configs {
    const INITURL    = 'https://rc.laborders.com';

    // SELENIUM SERVER INFO
    const BROWSER    = 'internet explorer'; // firefox, chrome, internet
explorer
    const HOST       = 'localhost';
    const PORT       = 4444;
…
```

Código 4 – Excerto do ficheiro de configuração do *Selenium WebDriver*

O *script* é chamado no teste através do comando:

```
Runtime.getRuntime().exec("C:\selenium\download.exe");
```

O diagrama de sequência da **Figura 11** descreve o teste representativo desta funcionalidade. São criadas concessões na plataforma de encomendas *online* da *LabOrders* que definem qual a área do projeto, o investigador principal encarregue, datas de ínicio e fim, bem como o orçamento disponível. Após ser criada uma concessão é possível gerar um documento .pdf com a sua informação. O teste irá gerar o documento de concessão e verificar se o seu conteúdo é o correto.

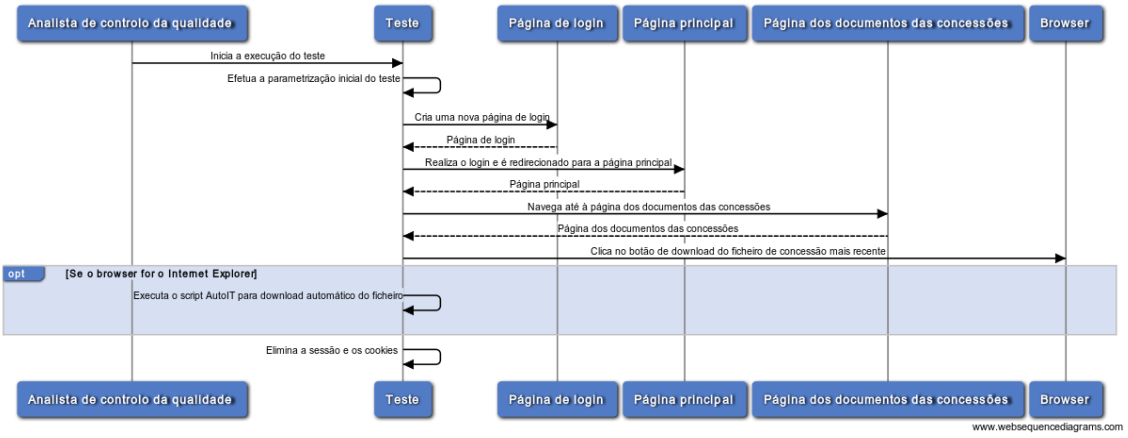

Figura 11 – Descrição do teste de *download* de um documento de concessão

O código pode ser consultado no anexo **"Teste de *Download* de um Documento de Concessão"**.

O problema do *download* automático de ficheiros através do *Internet Explorer* fica totalmente resolvido com a implementação desta solução. No entanto a *LabOrders* mudou de ideias após a implementação porque não queria ter o *Orbit* instalado apesar deste ser o gestor de *downloads* líder do mercado, gratuito e utilizado por milhões de pessoas. Desta forma a empresa preferiu não realizar transferências de ficheiros quando o *browser* a correr os testes fosse o *Internet Explorer*, tendo solicitado nestes casos que os testes fossem marcados como incompletos e sinalizados de forma a se distinguirem dos restantes. Tal foi conseguido seguindo o procedimento descrito no próximo capítulo.

3.4.1 Marcação de Testes com *Downloads* como Incompletos quando Executados através do *Internet Explorer*

Os métodos de teste vazios são interpretados como bem-sucedidos pelo *PHPUnit*. Esta interpretação errada leva a que os relatórios de testes se tornem inúteis porque não se consegue apurar se um teste é na verdade bem-sucedido ou se apenas ainda não foi implementado. Como tal não se pode ter uma instrução condicional que simplesmente ignore o conteúdo do teste caso o *browser* seja o *Internet Explorer*. Invocar `$this->fail()` no método não implementado (ou que se pretende ignorar) também não ajuda, já que o teste seria interpretado como se tivesse falhado. Isto seria tão errado como interpretar um teste não implementado como se fosse um sucesso. Associando um teste bem-sucedido a uma barra verde e um teste que falha a uma barra vermelha, chega-se à conclusão que é necessária uma cor adicional, amarelo por exemplo, para marcar um teste como incompleto ou ainda não implementado. É preciso sinalizar o teste como incompleto para que não seja interpretado como bem-sucedido:

```
class SampleTest extends PHPUnit_Framework_TestCase {
    public function testSomething() {
        // Optional: Test anything here, if you want
        $this->assertTrue(TRUE, 'This should already work');

        // Stop here and mark this test as incomplete
        $this->markTestIncomplete('This test has not been implemented yet');
    }
}
```

Código 5 – Exemplo de marcação de um teste como incompleto

Um teste incompleto distingue-se por ter um I (de *Incomplete*) no *output* da consola, conforme é mostrado no exemplo seguinte:

```
phpunit –verbose SampleTest
PHPUnit 4.5.0 by Sebastian Bergmann and contributors.

I

Time: 0 seconds, Memory: 3.95Mb

There was 1 incomplete test:

1)  SampleTest::testSomething
2)  This test has not been implemented yet
/home/sb/SampleTest.php:12
OK, but incomplete or skipped tests!
Tests: 1, Assertions: 1, Incomplete: 1.
```

Isto funciona bem na consola contudo o *NetBeans* não mostra a barra amarela na GUI, o erro já foi reportado e fechado como *WONTFIX*[3] porque o *NetBeans* interpreta o relatório XML do *PHPUnit* e apenas testes bem-sucedidos/erros são lá incluídos. Os testes ignorados (*skipped*) e incompletos não estão a ser incluídos. Enquanto o *PHPUnit* não resolver este problema é impossível mostrar a barra amarela na GUI do *NetBeans* (o *NetBeans* apenas reabrirá o *issue* depois do erro ter sido corrigido no *PHPUnit*). No entanto continua a ser recomendado marcar os testes não implementados como incompletos porque são mostrados como ignorados (barra vermelha) na GUI do *NetBeans* em vez de bem-sucedidos (barra verde). Os resultados restantes são exibidos corretamente.

[3] https://netbeans.org/bugzilla/show_bug.cgi?id=176157 [último acesso: Out 2015]

3.5 Exibição de Mensagens de Erro Personalizadas

A *LabOrders* tinha interesse em personalizar as mensagens de erro de forma a identificar mais facilmente os elementos *web* que a *framework* não conseguia encontrar. O *Selenium WebDriver* retorna apenas uma mensagem de erro genérica *"Element not found"*, sem qualquer identificador do elemento em questão, pelo que era necessário analisar o código sempre que um erro deste tipo surgia para apurar de que elemento se tratava.

A funcionalidade para a pesquisa de um único elemento [método `findElementBy()`] já tinha sido desenvolvida antes do projeto de mestrado ter tido início. No entanto, não suportava a procura de múltiplos elementos em simultâneo [método `findElementsBy()`] devido ao facto de esta se tratar de uma solução mais complexa que requeria a reescrita de várias funções. Como tal a exibição de mensagens de erro personalizadas no método `findElementsBy()` foi a segunda funcionalidade criada. O problema foi resolvido através da modificação dos PHP *bindings* do *Selenium WebDriver*, mais propriamente da classe ***WebDriverBase***, conforme se pode observar no seguinte excerto de código:

```
…

class WebDriverBase {
    protected $requestURL;
    protected $_curl;

    function __construct($_seleniumUrl) {
        $this->requestURL = $_seleniumUrl;
    }
    protected function &curlInit($url) {
        if($this->_curl === null) {
            $this->_curl = curl_init($url);
        }
        else {
            curl_setopt($this->_curl, CURLOPT_HTTPGET, true);
            curl_setopt($this->_curl, CURLOPT_URL, $url);
        }

        curl_setopt($this->_curl, CURLOPT_HTTPHEADER,
array("application/json;charset=UTF-8"));
        curl_setopt($this->_curl, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($this->_curl, CURLOPT_FOLLOWLOCATION, true);
        curl_setopt($this->_curl, CURLOPT_HEADER, false);
        return $this->_curl;
    }

    …

    /**
     * The Function analyses the status attribute of the response
     * For some statuses it throws an exception (for example
NoSuchElementException)
     * @param string $json_response
     * @param string $msg_error Error message thrown in NoSuchElementException
     */
    protected function handleResponse($json_response, $msg_error = null) {
        $status = $json_response->{'status'};
```

```
            switch ($status) {
                case WebDriverResponseStatus::Success:
                    return;
                    break;
                case WebDriverResponseStatus::NoSuchElement:
                    throw new NoSuchElementException($json_response, $msg_error);
                    break;
                default:
                    print_r($json_response);
                    throw new WebDriverException($status, 99, null);
                    break;
            }
        }

        …

        /**
         * Searches for multiple elements on the page, starting from the document
    root
         * @param string $locatorStrategy
         * @param string $value
         * @param string $msg_error Error message thrown in NoSuchElementException
         * @return array of WebElement
         */
        public function findElementsBy($locatorStrategy, $value, $msg_error = null)
    {
            $request = $this->requestURL . "/elements";
            $session = $this->curlInit($request);
            $args = array('using' => $locatorStrategy, 'value' => $value);
            $postargs = json_encode($args);
            $this->preparePOST($session, $postargs);
            $response = trim(curl_exec($session));
            $json_response = json_decode($response);
            $elements = $json_response->{'value'};
            $webelements = array();

            foreach ($elements as $key => $element) {
                $webelements[] = new WebElement($this, $element, null);
            }

            $size = sizeof($webelements);

            if($size == 0)
                throw new NoSuchElementException($json_response, $msg_error);

            return $webelements;
        }

        …

    }
```

Código 6 – Alterações à classe *WebDriverBase* do *Selenium WebDriver* para a exibição de mensagens de erro personalizadas

Foi incluído no método `findElementsBy()` o parâmetro `$msg_error` que é a mensagem de erro personalizada caso não seja encontrado o elemento *web* pretendido. Se o tamanho do vetor `$webelements[]` for igual a 0 significa que não foi possível localizar nenhum elemento na página usando os parâmetros fornecidos, como tal irá ser lançada uma nova exceção *NoSuchElementException* que também irá receber o parâmetro `$msg_error`.

Isto implica alterar a classe ***NoSuchElementException***:

```
class NoSuchElementException extends WebDriverException {
    private $json_response;

    public function __construct($json_response, $msg_error) {
        parent::__construct($msg_error, WebDriverResponseStatus::NoSuchElement,
null);
        $this->json_response = $json_response;
    }
}
```

Código 7 – Alterações à classe *NoSuchElementException* do *Selenium WebDriver* para a exibição de mensagens de erro personalizadas

Passando o parâmetro `$msg_error` para o construtor da classe ***NoSuchElementException*** garante-se que esse será o texto de erro exibido no *output* da consola. No entanto o uso desta funcionalidade não é obrigatório, o parâmetro `$msg_error` é opcional. Se não for preenchido no método `findElementsBy()` a mensagem de erro será *"Element not found"*. Se for preenchido a mensagem de erro tomará o seu valor. Por exemplo:

```
// Gets the Budget td's
$budgetTds = $budgetChildRow->findElementsBy(LocatorStrategy::tagName, $this-
>budgetTdTagName, "Could not find the Budget td's");
```

Código 8 – Exemplo de definição de uma mensagem de erro personalizada

Caso os *table data's* (td's, dados da tabela [colunas de uma tabela em HTML]) do orçamento (*budget)* não sejam encontrados, a mensagem de erro será: *"Could not find the Budget td's"*.

Com esta solução passa a ser possível incluir mensagens de erro personalizadas, não só na pesquisa de um único elemento mas também na procura de múltiplos elementos em simultâneo.

3.6 Análise do Conteúdo de Ficheiros .pdf

A terceira funcionalidade foi a análise do conteúdo de ficheiros .pdf que é de extrema importância porque só assim é possível verificar se as ordens de encomenda e os respetivos avisos de receção possuem o nome do produto, o nome do fornecedor e o preço corretos. Para isso foi utilizada a biblioteca *PD2Text* . A sua instalação realiza-se da seguinte forma:

1. Efetuar o *download* do *PDF2Text*: https://gist.github.com/neko-fire/7038322.
2. Extrair o conteúdo do ficheiro .zip transferido.
3. Criar o diretório *"C:\laborders\alpha.seleniumtests\lib\pdf2text"* e colocar lá o ficheiro *pdf2text.php*.

O método `pdf2text()` obtém o conteúdo do ficheiro .pdf em texto simples (sem formatação do .pdf) e de seguida são realizadas asserções sobre esse mesmo conteúdo:

```
// Gets the .pdf file content
$pdfContent = pdf2text($directory);

// Verifies if the Vendor name exists in the .pdf file
$this->assertContains($vendorName, $pdfContent, "The generated .pdf file
doesn't contain the Vendor name: " . $vendorName);
…
```

Código 9 – Exemplo da obtenção do conteúdo de um ficheiro .pdf e asserção sobre o seu conteúdo

O diagrama de sequência da **Figura 12** descreve o teste representativo desta funcionalidade. Após ser feita uma encomenda é possível gerar um documento .pdf que consiste na ordem de encomenda e contém o nome do fornecedor, o nome do produto e o preço total. O teste irá gerar a a ordem de encomenda e verificar se o seu conteúdo é o correto.

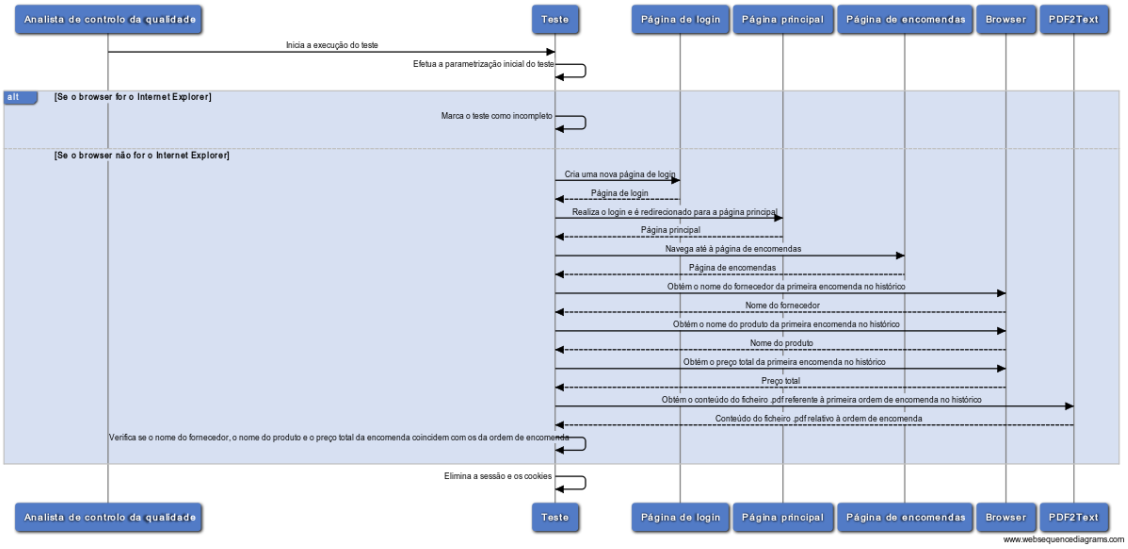

Figura 12 – Descrição do teste do conteúdo de uma ordem de encomenda

O código pode ser consultado no anexo **"Teste do Conteúdo de Uma Ordem de Encomenda".**

O desenvolvimento desta funcionalidade permite à empresa analisar não só o conteúdo dos ficheiros .pdf referentes às ordens de encomenda e respetivos avisos de recepção, como de quaisquer ficheiros .pdf.

3.7 Obtenção de Elementos *Web* através de Código *jQuery*

A versão 1.0 do *Selenium* (RC – *Remote Control*) utilizava extensivamente o *Sizzle* (motor de seletores CSS do *jQuery*) mas no *Selenium 2.0* (*WebDriver*) os programadores decidiram confiar na implementação de CSS dos *browsers* (estritamente CSS padrão) e apenas injetar o *Sizzle* quando não são suportados seletores de CSS nativos para o *JavaScript* (por exemplo quando se usa o *Internet Explorer 6/7* ou o *Firefox 3.0*).

Como tal, apesar de ser possível utilizar os seletores do *Sizzle* nos testes, esta é uma prática extremamente desaconselhada porque só irá funcionar num conjunto reduzido de *browsers*, em versões antigas e desatualizadas [Wilson, 2011]. Para além disso os seletores do *Selenium* permitem encontrar os mesmos elementos que os do *Sizzle* e, na maioria dos casos, de uma forma mais simples e imediata. No entanto, como a *LabOrders* fazia questão que a *framework* suportasse a obtenção de elementos *web* e respetivos atributos através de código *jQuery*, foi realizada uma pesquisa através da qual se chegou à conclusão que tal é possível através do método `executeScript()` [Bukowicz, 2012]. No caso dos atributos é direto, basta passar o código *jQuery* como parâmetro ao método `executeScript()` para este retornar a informação pretendida:

```
// Finds the user Row and retrieves the Row ID attribute
$rowId = $this->webdriver->executeScript('return $("tr:contains(\'' . $email .
'\')").attr(\'ID\')', array());
```

Código 10 – Exemplo da obtenção de um atributo de um elemento *web* através de código *jQuery*

O código anterior obtém a *table row* (tr – linha de uma tabela em HTML) que contém o *e-mail* `$email` e retorna o atributo `ID` dessa mesma linha.

No caso dos elementos *web* é mais complicado porque o `executeScript()` retorna um objeto do tipo ***stdClass***, uma classe predefinida e genérica do PHP que é vazia, ou seja, não possui métodos nem atributos. Normalmente é usada quando se converte para objeto outros tipos de dados [Stack Overflow, 2009]. Também é útil para definir propriedades dinâmicas, conforme se irá verificar mais adiante, bem como criar objetos e atribuir-lhes dados sem ter que definir formalmente uma classe. Para o objeto poder ser manipulado pelo *Selenium* é preciso convertê-lo de ***stdClass*** para ***WebElement***, o que foi conseguido recorrendo à técnica de *Reflection* do PHP, como já foi mencionado no subcapítulo **"Comparação e Escolha da Ferramenta de Testes Funcionais"** do capítulo **"Metodologias de Desenvolvimento e Testes Automáticos de *Software*"**. A *Reflection* (reflexão) ocorre quando um objeto é capaz de se examinar a si próprio, informando o programador dos seus métodos e propriedades em tempo de execução:

```
/**
 * Class casting
 *
 * @param string|object $destination
 * @param object $sourceObject
 * @return object
 */
function cast($destination, $sourceObject) {
    if(is_string($destination)) {
        $destination = new $destination();
    }
    $sourceReflection = new ReflectionObject($sourceObject);
    $destinationReflection = new ReflectionObject($destination);
    $sourceProperties = $sourceReflection->getProperties();
    foreach ($sourceProperties as $sourceProperty) {
        $sourceProperty->setAccessible(true);
        $name = $sourceProperty->getName();
        $value = $sourceProperty->getValue($sourceObject);
        if ($destinationReflection->hasProperty($name)) {
            $propDest = $destinationReflection->getProperty($name);
            $propDest->setAccessible(true);
            $propDest->setValue($destination,$value);
        }
        else {
            $destination->$name = $value;
        }
    }

    return $destination;
}
```

Código 11 – Função que converte um objeto *stdClass* para *WebElement*

Resumidamente, no código acima exposto é criado o objeto de destino (`$destination`), que no final do processo de *Reflection* irá ter o tipo pretendido com todas as propriedades do objeto fonte (`$sourceObject`). São também criados dois *ReflectionObjects* (clones), um do objeto fonte e outro do objeto de destino, para realizar a conversão. Visto que a ***stdClass*** permite que as instâncias dos objetos tenham propriedades dinâmicas como foi referido anteriormente, estes *ReflectionObjects* são necessários para não alterar as propriedades dos objetos originais em tempo de execução, o que poderia afetar o resultado final.

De seguida são obtidas, iteradas e definidas como acessíveis as propriedades do objeto fonte, a partir das quais se consegue aceder aos nomes e valores respetivos. Finalmente, são criadas essas propriedades com os dados corretos no objeto de destino que é retornado com o tipo desejado.

De forma a resolver o problema, a função `cast()` vai ser invocada nos testes sempre que se pretender obter elementos *web* através de código *jQuery* com o objetivo de converter o objeto ***stdClass*** retornado pelo método `executeScript()` para ***WebElement***. O primeiro parâmetro da função `cast()` é a *string* (cadeia de carateres) referente ao tipo de dados do objeto de destino, ou seja `'WebElement'`, e o segundo parâmetro é o objeto ***stdClass*** obtido através do método `executeScript()`:

```
class grantsDocsPage extends basePage {
    …

    public function uploadGrantDoc() {
        // Gets the grantsDocsFileInput in stdClass type and stores it in the
$grantsDocsfileInputStdClass variable
        $grantsDocsfileInputStdClass = $this->webdriver->executeScript('return
$("#itemlist .grant_file_upload:first")', array());

        // Casts $grantsDocsfileInputStdClass to WebElement and stores it in
the $grantsDocsfileInput variable
        $grantsDocsfileInput = $this->cast('WebElement', $grantsDocsfileInput);
        …
    }

    …

}
```

Código 12 – Exemplo da utilização da função *cast()* nos testes

A implementação desta funcionalidade permite à *framework* obter todo o tipo de elementos *web* e respetivos atributos, tanto através dos seletores do *Selenium* como de código *jQuery*.

3.8 Correção da Funcionalidade de Envio do Relatório de Testes por Correio Eletrónico

A *LabOrders* já tinha desenvolvido a funcionalidade de envio do relatório de erros por correio eletrónico antes do projeto de mestrado ter tido início mas como não se encontrava a funcionar corretamente foi necessário proceder a várias alterações que irão ser descritas de seguida. O primeiro problema identificado foi o facto dos resultados de alguns testes não serem incluídos no relatório e já foi abordado previamente no subcapítulo **"Marcação de Testes com *Downloads* como Incompletos quando Executados através do *Internet Explorer*"**, devendo-se a um erro do *PHPUnit*. O segundo problema consiste na mensagem de correio eletrónico não ser enviada porque as extensões do PHP requeridas se encontravam desativadas. Foi resolvido através da ativação do *php_imap* e do *php_openssl* no ficheiro *"C:\wamp\bin\php\php5.3.4\php.ini"* removendo o caráter *';'* que se encontra no início das linhas:

```
;extension=php_imap.dll
;extension=php_openssl.dll
```

Código 13 – Excerto do ficheiro *php.ini*

O terceiro e último problema reside na mensagem de correio eletrónico não conter o ficheiro comprimido de anexo com as capturas de ecrã dos testes. Foi solucionado com a instalação e inclusão do *WinRAR*, um compactador e descompactador de dados, na lista de *software* autorizado da empresa. Os diretórios das dependências também tiveram de ser alterados. Segue-se um excerto do código modificado na classe auxiliar *email*:

```
/**
 * Description of email
 *
 * Provides an abstraction layer to validate the e-mails sent through the tests
 */

require_once '../lib/PHPMailer_5.2.4/class.phpmailer.php';

class email {
    …

    /**
     * Compresses the screenshot folder from the current day
     */
    private static function _zipTodaysScreenshotFolder() {
        // Initializes the zip file path
        $zipFilePath = NULL;

        // Gets today's year, month and day
        $year = date('Y');
        $month  = date('m');
        $day = date('d');

        /* @var string $path Path to the screenshots directory */
        $path = getcwd() . "\\..\\error_screenshot\\$year\\$month\\$day";
```

```
        /* @var string $destination Compressed file destination */
        $destination = getcwd() . "\\..\\error_screenshot\\imagesLog.rar";

        /* @var string $winRARPath Path to WinRAR */
        $winRARPath    = 'C:\"Program Files"\WinRAR\Rar.exe';

        // Checks if a previous compressed file exists and if so deletes it
        if(file_exists($destination)) {
            // Deletes the previous compressed file
            unlink($destination);
        }

        if(file_exists($path)) {
            // Compresses the folder
            exec('cd ' . $path . ' & ' . $winrarPath . ' a -r ' . $destination .
' "*.*"');

            if(!file_exists($destination)) {
                throw new Exception("Could not create the tests screenshot
attachment");
            }

            // Stores the destination path in the $zipFilePath variable
            $zipFilePath = $destination;

        }

        // Returns the $zipFilePath variable
        return $zipFilePath;
    }

    …
}
```

Código 14 – Alterações à classe *email* para o envio da mensagem de correio eletrónico com os resultados dos testes

O código anterior cria um ficheiro comprimido, utilizando o *WinRAR,* com as capturas de ecrã dos testes do dia atual. O ficheiro é criado no diretório *"\error_screenshot\"* no mesmo nível do diretório atual de trabalho. Previamente não estava a ser criado porque o *WinRAR* não estava instalado e o seu diretório no código estava errado. A instalação do *WinRAR* e a correção do caminho para o seu ficheiro executável fizeram com que todas as mensagens de correio eletrónico contendo os relatórios dos testes fossem enviadas tendo como anexo o ficheiro comprimido que inclui as capturas de ecrã.

3.9 Desenvolvimento de Testes Funcionais

Foram codificadas dezenas de testes funcionais como trabalho suplementar, uma vez que apenas faziam parte dos objetivos deste projeto os testes demonstrativos das novas funcionalidades implementadas. A finalidade destes testes é apurar o funcionamento correto da plataforma de encomendas *online* da *LabOrders*, potenciando assim uma melhor classificação final na dissertação. Por motivos de limitação de espaço não é possível incluir neste documento sequer 5% do código desenvolvido, no entanto é importante referir que foram criados testes dos:

- **Serviços, pesquisa de produtos e registo de utilizadores (instituições) [https://www.youtube.com/watch?v=Otm4nzY3vNM]**: este teste irá ser detalhado posteriormente neste subcapítulo, como exemplo, utilizando diagramas de sequência.
- **Criação de utilizadores de contabilidade [https://www.youtube.com/watch?v=tDf5YaThJVs]**: o teste realiza o *login* como administrador, navega até à página de criação de utilizadores de contabilidade e preenche os campos referentes ao departamento de contabilidade, nome e *e-mail* do utilizador que pretende criar. Após clicar no botão para Guardar (*Save*) a informação verifica se a mensagem de sucesso foi exibida.
- **Adição, edição, ativação e desativação das concessões, pelas direções das instituições e pelos laboratórios (clientes da plataforma) [https://www.youtube.com/watch?v=l4nJcgUIvkg]**:
    - **[0:00 – 0:27]: Adição de uma concessão pela direção de uma instituição**
      O teste efetua o *login* como a administração do Instituto de Biologia Experimental e Tecnológica (iBET), navega até à página das concessões (*Grants*) na plataforma, clica no botão adicionar concessão (*Add grant*) e preenche os campos relativos ao nome, PI e ID da concessão que deseja criar, bem como os privilégios dos utilizadores de contabilidade que lhe estão associados. Após clicar no botão para submeter (*Submit*) a informação confirma se a mensagem de sucesso foi mostrada. Finalmente verifica se o número de concessões criadas foi incrementado em uma unidade.
    - **[0:27 – 0:43]: Adição de uma concessão por um laboratório**
      O teste realiza o login como o laboratório *Decantum*, navega até à página das concessões na plataforma, clica no botão adicionar concessão e preenche o nome da concessão que pretende criar. Após clicar no botão para submeter a informação verifica se a mensagem de sucesso foi exibida. Por fim confirma se o número de concessões criadas foi incrementado em uma unidade.
    - **[0:43 – 1:07]: Edição de uma concessão pela direção de uma instituição**
      O teste efetua o *login* como a administração do iBET, navega até à página das concessões na plataforma, clica no botão de edição da última concessão criada na tabela, na coluna de ações (*Actions*) e altera o nome dessa concessão. Após clicar no botão para submeter a informação verifica se a mensagem de sucesso foi mostrada.

- **[1:07 – 1:22]: Edição de uma concessão por um laboratório**
  Este teste é igual ao anterior com a particularidade de realizar o *login* como o laboratório *Decantum*.
- **[1:22 – 2:05]: Desativação de uma concessão pela direção de uma instituição**
  O teste efetua o *login* como a administração do iBET, navega até à página das concessões na plataforma, clica no botão de eventos da última concessão criada na tabela, na coluna de ações e confirma se essa concessão se encontra ativa (*Active*) na coluna de estado (*Status*) da tabela de eventos. De seguida clica no botão de edição dessa mesma concessão e desativa a *checkbox* que define se está ativa ou não. Após clicar no botão para submeter a informação verifica se a mensagem de sucesso foi exibida. O penúltimo passo é voltar a clicar no botão de eventos para confirmar se o estado da concessão se alterou para desativada (*Disabled*). Finalmente verifica se o número de concessões desativadas foi incrementado em uma unidade.
- **[2:05 – 2:50]: Ativação de uma concessão pela direção de uma instituição**
  Este teste é o oposto do anterior, ou seja em vez de desativar a *checkbox* que define se a concessão está ativa ou não, ativa-a, confirma se o estado da concessão se alterou de desativada para ativa e verifica se o número de concessões ativas foi incrementado em uma unidade.
- **[2:50 – 3:26]: Desativação de uma concessão por um laboratório**
  Este teste é igual ao teste respetivo da direção de uma instituição com a particularidade de realizar o *login* como o laboratório *Decantum*.
- **[3:26 – 4:07]: Ativação de uma concessão por um laboratório**
  Tal como o anterior, este teste é equivalente ao teste respetivo da direção de uma instituição com a particularidade de efetuar o *login* como o laboratório *Decantum*.

- **U*pload*, *download* e eliminação dos documentos das concessões na plataforma [https://www.youtube.com/watch?v=ZPeEilnDcFY]**:
  - **[0:00 – 0:17]: *Upload* de um documento de concessão na plataforma**
    O teste realiza o *login* como PI do laboratório do iBet, navega até à página dos documentos das concessões (*Grants' Docs*) na plataforma, clica no botão navegar (*Browse*), seleciona o ficheiro que deseja enviar, submete a informação da janela de navegação e confirma se a mensagem de sucesso foi mostrada. Por fim verifica se o número de documento de concessão na plataforma foi incrementado em uma unidade.
  - **[0:17 – 0:32]: *Download* de um documento de concessão na plataforma**
    O teste efetua o *login* como PI do laboratório do iBet, navega até à página dos documentos das concessões na plataforma, clica no botão de *download* do ficheiro na coluna de ações da tabela e confirma se o ficheiro foi transferido corretamente para o diretório de *download*.

- **[0:32 – 0:49]: Eliminação de um documento de concessão na plataforma**
  O teste realiza o *login* como PI do laboratório do iBet, navega até à página dos documentos das concessões na plataforma, clica no botão de eliminação do ficheiro na coluna de ações da tabela e verifica se a mensagem de sucesso foi exibida. Finalmente confirma se o número de documentos de concessão na plataforma foi decrementado em uma unidade.

- **Importação de catálogos**
  [https://www.youtube.com/watch?v=M8uc_5lQfHg]: o teste efetua o *login* como PI do laboratório do iBET, pesquisa pela referência de um dos produtos que foram importados e verifica se esse produto existe na plataforma.
- **Alteração de preços durante as requisições**
  [https://www.youtube.com/watch?v=8Cq1h2mC5HA]:
  - **[0:00 – 0:58]: Requisitar um produto começando por não alterar o seu preço**
    O teste realiza o *login* como PI do laboratório do iBET, pesquisa pela referência de um produto e clica na hiperligação para requisitar (*Request*) esse produto, na coluna de ações da tabela. Não altera o preço por unidade (*Price per unit*) do produto e após clicar no botão para submeter a informação, confirma se a mensagem de sucesso foi mostrada. Como o preço original do catálogo não foi alterado, deve aparecer um ícone circular verde ao lado esquerdo do preço. O teste verifica se esse ícone é exibido e de seguida clica na hiperligação para abrir (*Open*) a requisição, na coluna de ações da tabela, e na hiperligação de edição (*Edit*) da coluna de ações da tabela de resumo da requisição (*Summary of requisition*). Seguidamente clica no botão para modificar (*change*) o preço por unidade e após clicar no botão para guardar a informação confirma se a mensagem de sucesso foi exibida. Como o preço foi alterado para um valor diferente do que estava originalmente no catálogo, deve aparecer um ícone triangular vermelho, que sinaliza essa situação, ao lado esquerdo do preço. O teste verifica se esse ícone é mostrado, volta a clicar na hiperligação para abrir a requisição e preenche os campos concessão de financiamento (*Funding grant*), rubrica (*Rubric*), subrubrica (*Subrubric*) e endereço de envio (*Shipping address*) para a aprovação da requisição. Após clicar no botão para submeter a informação, confirma se a mensagem de sucesso foi exibida e faz *logout*. O próximo passo é efetuar o *login* como um utilizador do departamento de compras do iBET e clicar no botão para abrir a requisição, seguido do botão para encomendar manualmente (*Order manually by myself*), de forma a confirmar a requisição. Após clicar no botão verifica se a mensagem de sucesso foi mostrada e faz *logout*. O penúltimo passo é realizar o *login* como o fornecedor GRiSP, pesquisar pela referência do produto e alterar o preço (*Price*) de volta para o que estava originalmente no catálogo, na coluna respetiva da tabela. Isto deverá fazer com que o ícone triangular vermelho seja substituído pelo ícone circular verde na requisição. Por fim, após fazer *logout* volta a efetuar o *login* como PI do laboratório do

iBET de forma a verificar se o ícone verde está a ser exibido ao lado esquerdo do preço da última requisição (mais recente) da tabela.

- **[0:58 – 2:02]: Requisitar um produto começando por alterar o seu preço**
  Este teste é o oposto do anterior, ou seja, enquanto PI do laboratório do iBET começa por requisitar o produto alterando o seu preço e confirma se o ícone triangular vermelho é mostrado. De seguida abre a requisição, volta a alterar o preço para o valor que estava originalmente no catálogo e verifica se ícone circular verde é exibido. Após aprovação e confirmação da requisição, o teste enquanto fornecedor GRiSP pesquisa pelo produto e altera o preço uma vez mais. Finalmente volta a realizar o *login* como PI do laboratório do iBET de forma a verificar se o ícone triangular vermelho está a ser mostrado.

- ***Download* das ordens de encomenda e respetivos avisos de recepção [https://www.youtube.com/watch?v=H1UFiDbL6Fs]**:
  - **[0:00 – 0:48]: *Download* do aviso de receção**
    O teste efetua o *login* como PI do laboratório do iBET, pesquisa pela referência de um produto e clica na hiperligação para requisitar esse produto, na coluna de ações da tabela. Após clicar no botão para submeter a informação, confirma se a mensagem de sucesso foi exibida. De seguida clica na hiperligação para abrir a requisição e preenche os campos custo dos portes de envio (*Shipping fee*), concessão de financiamento, rubrica, subrubrica e endereço de envio para a aprovação da requisição. Após clicar no botão para submeter a informação, verifica se a mensagem de sucesso foi mostrada e faz *logout*. Seguidamente realiza o *login* como um utilizador do departamento de compras do iBET e clica no botão para abrir a requisição, seguido do botão para encomendar manualmente, de forma a confirmar a requisição. O próximo passo é clicar na hiperligação de avisos de receção (*Receivals*) na coluna de ações da tabela, de forma a obter o nome do fornecedor, o nome do produto e o preço. O penúltimo passo é clicar no botão de *download* do ficheiro .pdf referente ao aviso de receção na coluna de ações da tabela e verificar se este foi transferido corretamente para o diretório de *download*. Por fim o teste verifica se o ficheiro .pdf contém o nome do fornecedor, o nome do produto e o preço corretos (obtidos anteriormente).
  - **[0:48 – 1:18]: *Download* da ordem de encomenda**
    O teste efetua o *login* como PI do laboratório do iBET, navega até à página de encomendas (*Lab Orders*) e obtém o nome do fornecedor, o nome do produto e o preço da primeira encomenda (mais recente) na tabela. De seguida seleciona essa encomenda e clica no botão PDF para fazer o *download* do ficheiro .pdf relativo à ordem de encomenda. Finalmente o teste verifica se o ficheiro foi transferido corretamente para o diretório de *download* e se contém o nome do fornecedor, nome do produto e preço corretos (obtidos anteriormente).

- **Criação, edição, desativação e ativação de calendários [https://www.youtube.com/watch?v=XIwRxDTA3tw]**:
    - **[0:00 – 0:30]: Criação de um calendário**
      O teste realiza o *login* como a administração do iBET, navega até à página de calendários (*Timesheets*), escolhe o utilizador ao qual pretende adicionar um calendário e clica no botão respetivo (*Add timesheet*). Preenche os campos referentes à concessão, duração total (*Total duration*) e número máximo de horas por dia (*Max. hours per day*) do calendário que deseja criar. Após clicar no botão para guardar a informação confirma se a mensagem de sucesso foi exibida. De seguida clica no botão para adicionar um pacote de trabalho (*work package*) na coluna de ações do primeiro calendário (mais recente) na tabela e preenche os campos relativos ao nome (*Name*) e duração (*Duration*) do pacote de trabalho que pretende criar. O penúltimo passo é clicar no botão para guardar a informação e confirmar se a mensagem de sucesso foi mostrada. Por fim verifica se o número de calendários foi incrementado em uma unidade.
    - **[0:30 – 0:48]: Edição de um calendário**
      O teste efetua o *login* como a administração do iBET, navega até à página de calendários, seleciona o utilizador do qual deseja editar um calendário e clica no botão editar calendário, na coluna de ações do primeiro calendário (mais recente) na tabela. De seguida modifica os valores da duração total e do número máximo de horas por dia do calendário, clica no botão para guardar a informação e confirma se a mensagem de sucesso foi exibida.
    - **[0:48 – 1:09]: Desativação de um calendário**
      O teste realiza o *login* como a administração do iBET, navega até à página de calendários, escolhe o utilizador do qual pretende desativar um calendário e clica no botão para submeter a informação. De seguida clica no botão para desativar um calendário na coluna de ações do primeiro calendário (mais recente) ativo na tabela e verifica se a mensagem de sucesso foi mostrada. Finalmente confirma se o número de calendários desativados foi incrementado em uma unidade.
    - **[01:09 – 1:32]: Ativação de um calendário**
      Este teste é o oposto do anterior, ou seja, enquanto administração do iBET navega até à página de calendários, seleciona o utilizador do qual deseja ativar um calendário e clica no botão para submeter a informação. De seguida clica no botão para ativar um calendário na coluna de ações do primeiro calendário (mais recente) desativado na tabela e confirma se a mensagem de sucesso foi exibida. Por fim verifica se o número de calendários ativos foi incrementado em uma unidade.

- **Adição, edição, desativação e ativação de feriados [https://www.youtube.com/watch?v=hh1bezKJs48]**:
    - **[0:00 – 0:31]: Adição de um feriado**
      O teste efetua o *login* como a administração do iBET, navega até à página de calendários, escolhe o utilizador em cujo calendário deseja adicionar um feriado e clica no botão para submeter a informação. De seguida clica no botão para adicionar um feriado (*Add statutory holiday*), preenche os campos referentes ao nome e data (*Date*) do feriado que pretende criar, clica no botão para guardar a informação e verifica se a mensagem de sucesso foi exibida. Finalmente confirma se o número de feriados foi incrementado em uma unidade.
    - **[0:31 – 0:53]: Edição de um feriado**
      O teste realiza o *login* como a administração do iBET, navega até à página de calendários, seleciona o utilizador em cujo calendário pretende editar um feriado e clica no botão para submeter a informação. De seguida clica no botão para editar um feriado na coluna de ações do último calendário (mais recente) ativo na tabela, altera os valores do nome e da data do feriado, clica no botão para guardar a informação e verifica se a mensagem de sucesso foi mostrada. Finalmente confirma se o nome e a data do feriado foram editados corretamente para os valores desejados.
    - **[0:53 – 1:24]: Desativação de um feriado**
      O teste efetua o *login* como a administração do iBET, navega até à página de calendários, escolhe o utilizador em cujo calendário pretende desativar um feriado e clica no botão para submeter a informação. De seguida clica no botão para desativar um feriado na coluna de ações do último calendário (mais recente) ativo na tabela e verifica se a mensagem de sucesso foi exibida. Finalmente confirma se o número de feriados desativados foi incrementado em uma unidade.
    - **[1:24 – 2:17]: Ativação de um feriado**
      Este teste é o oposto do anterior, ou seja, enquanto administração do ibet clica no botão para ativar um feriado na coluna de ações do último calendário (mais recente) desativado na tabela e verifica se a mensagem de sucesso foi mostrada. Finalmente confirma se o número de feriados ativos foi incrementado em uma unidade.

- **Criação e eliminação de tarefas, adição e eliminação de períodos de férias, criação de utilizadores (funcionários dos laboratórios), reposição de palavras-passe, edição das definições dos utilizadores e eliminação de datas de contrato [https://www.youtube.com/watch?v=imY0TijP56c]**:
    - **[0:00 – 0:22]: Criação de uma tarefa**
      O teste realiza o *login* como a administração do iBET, navega até à página de calendários, seleciona o utilizador em cujo calendário deseja criar uma tarefa e clica no botão para submeter a informação. De seguida clica no primeiro dia de trabalho do calendário, preenche os campos relativos à descrição (*Description*), pacote de trabalho e duração da tarefa. Após clicar no botão para guardar a informação verifica se a mensagem de sucesso foi exibida. Finalmente confirma se o número de tarefas foi incrementado em uma unidade.
    - **[0:22 – 0:39]: Eliminação de uma tarefa**
      O teste efetua o *login* como a administração do iBET, navega até à página de calendários, seleciona o utilizador em cujo calendário pretende eliminar uma tarefa e clica no botão para submeter a informação. O próximo passo é clicar na primeira tarefa do calendário, seguida do botão para eliminar a tarefa (*Delete this task*). Por fim verifica se a mensagem de sucesso foi mostrada e se o número de tarefas foi decrementado em uma unidade.
    - **[0:39 – 0:59]: Adição de um período de férias**
      O teste realiza o *login* como a administração do iBET, navega até à página de calendários, seleciona o utilizador a que pretende adicionar um período de férias e clica no botão para submeter a informação. De seguida clica no botão para gerir férias (*Manage vacations*) e preenche os campos referentes à data de início e à data de fim do período de férias. Após clicar no botão para guardar a informação confirma se a mensagem de sucesso foi exibida.
    - **[0:59 – 1:43]: Eliminação de um período de férias**
      O teste efetua o *login* como a administração do iBET, navega até à página de calendários, seleciona o utilizador a que pretende eliminar um período de férias e clica no botão para submeter a informação. O próximo passo é clicar no primeiro dia de férias do calendário, seguido do botão respetivo para o eliminar (*Delete this vacation day*). Finalmente verifica se a mensagem de sucesso foi mostrada e se o número de dias de férias foi decrementado em uma unidade.

- **[1:43 – 2:05]: Criação de um utilizador**
  O teste realiza o *login* como a administração do iBET, navega até à pagina de membros do laboratório (*Lab Members*), seleciona o PI do laboratório onde se pretende criar o utilizador e clica no botão para submeter a informação. De seguida clica no botão para adicionar um utilizador (*Add User*), preenche os campos relativos ao nome e *e-mail* do utilizador que deseja criar e clica no botão para guardar a informação. Por fim confirma se a mensagem de sucesso foi exibida, navega até à página de calendários e verifica se o novo utilizador se encontra na lista.
- **[2:05 – 2:33]: Reposição das palavras-passe**
  O teste efetua o *login* como administrador, navega até à página de utilizadores de teste (*Test Users*), preenche o campo relativo à palavra-passe (*Password*) que se pretende definir e clica no botão para submeter a informação.
- **[2:33 – 2:52]: Edição das definições de um utilizador**
  O teste realiza o *login* como a administração do iBET, navega até à pagina de calendários, seleciona o utilizador adicionado no teste "Criação de um utilizador" e clica no botão para submeter a informação. De seguida clica no botão para editar as definições do utilizador e preenche os campos referentes à data de início do contrato (*Contract start date*) e à data de fim do contrato (*Contract end date*). Após clicar no botão para submeter a informação confirma se a mensagem de sucesso foi mostrada. Finalmente volta a clicar no botão para editar as definições de utilizador de forma a verificar se as datas de contrato assumiram os valores corretos (definidos previamente).
- **[2:52 – 3:11]: Eliminação de datas de contrato**
  O teste realiza o *login* como a administração do iBET, navega até à pagina de calendários, seleciona o utilizador adicionado no teste "Criação de um utilizador" e clica no botão para submeter a informação. O próximo passo é clicar no botão para editar as definições do utilizador, seguido do botão para eliminar datas de contrato (*Delete contract dates*). Por fim confirma se a mensagem de sucesso foi exibida.

Incluem-se de seguida, como exemplo, os diagramas de sequência que descrevem o caso de teste dos serviços. A *LabOrders* utiliza o *Sphinx* (http://sphinxsearch.com/, ferramenta de análise quantitativa e qualitativa de dados) para efetuar as pesquisas. O teste irá procurar por um produto e verificar se são retornados resultados, ou seja, se o serviço *Sphinx* está a correr (**Figura 13**).

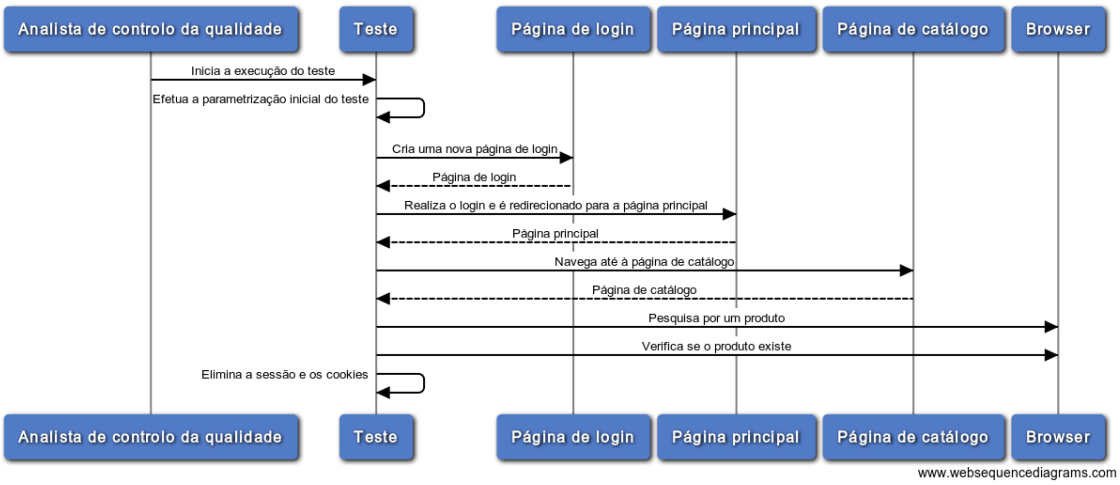

Figura 13 – Descrição do teste do serviço *Sphinx*

A empresa envia mensagens de correio eletrónico através da plataforma, por isso é preciso testar o *Simple Mail Transfer Protocol* (SMTP). Entende-se por SMTP um protocolo para enviar *e-mails* entre servidores. Numa primeira fase o registo (das instituições) na plataforma é feito através de uma mensagem de correio eletrónico enviada através do formulário do *site*. O teste irá preencher os campos desse formulário, submetê-lo e verificar se a mensagem de sucesso é mostrada, o que significa que o *e-mail* foi enviado e por conseguinte, que o serviço SMTP está a funcionar corretamente (**Figura 14**).

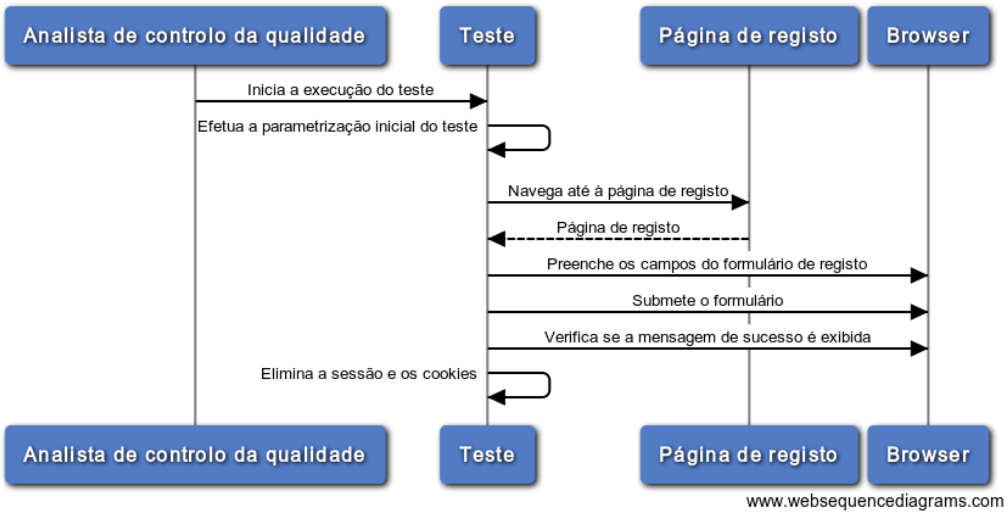

Figura 14 – Descrição do teste do serviço SMTP

O *SoapUI* (http://soapui.org/, solução *open source* de testes funcionais) é utilizado para simular as respostas dos serviços *System Application and Products* (SAP). Entende-se por SAP a empresa de *software* Alemã cujos produtos permitem acompanhar as interações dos negócios e dos clientes. Conforme se pode ver no vídeo https://youtu.be/Otm4nzY3vNM?t=37s, o teste realiza o *login* como PI do laboratório do Instituto Calouste Gulbenkian (IGC), navega até à página de orçamentos (*Budgets*) e expande todas as concessões (*Grants*). Se os gráficos das colunas de execução (*Execution*) e linha do tempo (*Timeline*) dos orçamentos dos projetos forem exibidos, isso significa que o serviço *SoapUI* está a correr uma vez que os gráficos são gerados a partir da resposta por si simulada (**Figura 15**). Em gestão de projetos entende-se por linha do tempo a linha horizontal que relaciona a conclusão de uma tarefa com o seu início.

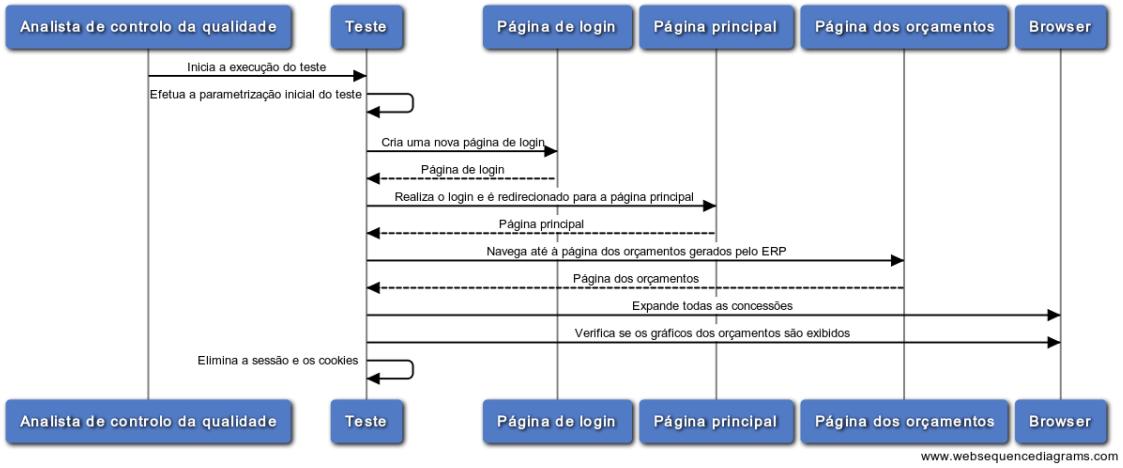

Figura 15 – Descrição do teste do serviço *SoapUI*

O código pode ser consultado no anexo **"Caso de Teste dos Serviços"**.

Foram também efetuadas várias correções nos testes existentes antes do estágio ter tido início, como por exemplo a substituição da função depreciada `getValue()` por `getAttribute('value')`, com o objetivo de eliminar os erros que estavam a ocorrer.

Concluído o primeiro projeto, a *framework* de testes funcionais e unitários sem dependências externas, iniciou-se o desenvolvimento da *framework* de testes unitários com dependências externas (base de dados e serviços) e respetiva prova de conceito, que irá ser relatado nos próximos capítulos.

3.10 Arquitetura da *Framework* de Testes Unitários Com Dependências Externas

O segundo projeto desenvolvido foi uma *framework* de testes unitários com dependências externas (base de dados e serviços). É composta por quatro componentes: três *frameworks* (*Kohana*, *Simpletest*, *PHPUnit*) e uma biblioteca (*Phactory*). Pode-se observar o diagrama de arquitetura da *framework* na **Figura 16**.

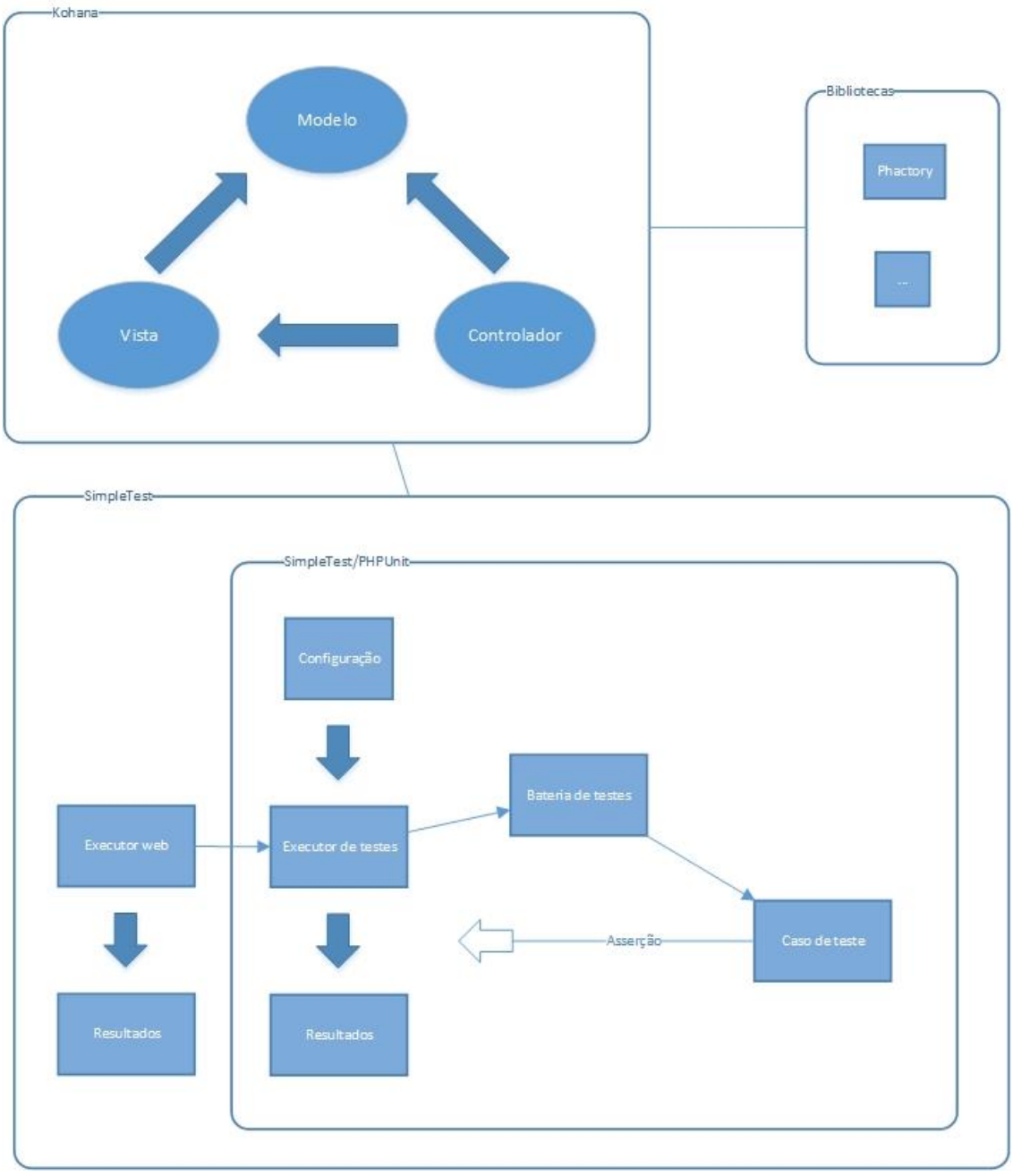

Figura 16 – Diagrama de arquitetura da *framework* de testes unitários sem dependências externas

O *Kohana* é utilizado pela *LabOrders* para desenvolver a sua plataforma *online* de encomendas, que é a aplicação que se pretende testar. Usa o *Model-View-Controller* (MVC), um padrão de arquitetura de *software* que separa a representação da informação e a *interface* com a qual o utilizador interage [Belem, 2013].

Propõe três componentes principais para serem usados no desenvolvimento de *software*:

- Os Modelos (*Models*): camada que representa os dados da aplicação e fornece meios de acesso (leitura e escrita) aos mesmos.
- As Vistas (*Views*): coleção de classes representativas dos elementos na *interface* do utilizador (tudo o que o utilizador pode ver no ecrã e com que pode interagir como botões, painéis de visualização, entre outros).
- Os Controladores (*Controllers*): classes que ligam os modelos e as vistas, e que são usadas para comunicar entre classes nos modelos e nas vistas.

Tanto o *SimpleTest* como o *PHPUnit* estão integrados na *framework*, podendo ambos ser utilizados para criar os testes unitários. No entanto, neste contexto é recomendado o uso do *SimpleTest* pelos motivos já apresentados no subcapítulo **"Comparação e Escolha da *Framework* de Testes Unitários"** do capítulo **"Metodologias de Desenvolvimento e Testes Automáticos de *Software*"**. A arquitetura do *SimpleTest* é em tudo semelhante à arquitetura do *PHPUnit* que já foi descrita no subcapítulo **"Arquitetura da *Framework* de Testes Funcionais e Unitários Sem Dependências"**, com a particularidade de possuir um executor *web* que permite visualizar os resultados dos casos de testes no *browser*.

A biblioteca *Phactory* é utilizada quando é necessário testar a integridade dos dados através da *mock database*.

Apresentada a arquitetura da *framework* de testes funcionais e unitários com dependências externas, os componentes que a constituem e respetivas funções, no capítulo seguinte irá ser descrita a estrutura desses mesmos componentes e a forma como se relacionam entre si.

3.11 Estrutura da *Framework* de Testes Unitários Com Dependências Externas

A estrutura da *framework* de testes unitários com dependências externas é representada através do diagrama da **Figura 17** que descreve a relação entre as classes dos componentes que a constituem. Trata-se de um diagrama de classes simplificado porque as bibliotecas/*frameworks* utilizadas possuem centenas de classes e como tal apenas foram incluídas aquelas com que se trabalhou diretamente.

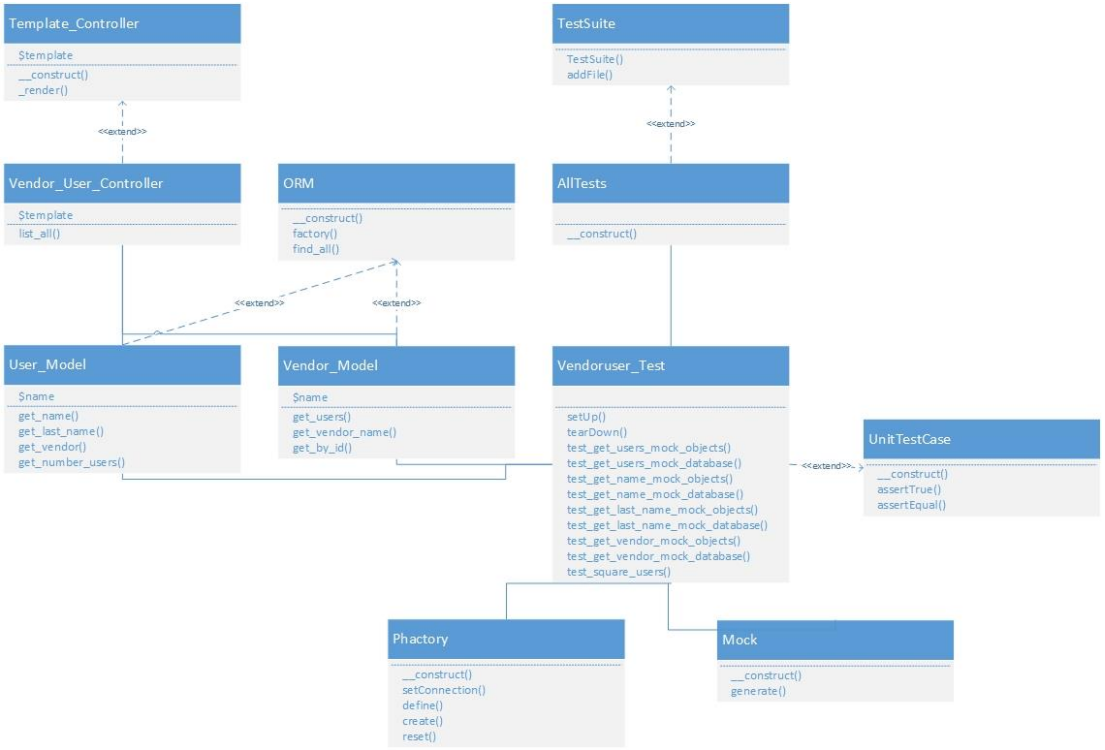

Figura 17 – Diagrama de classes da *framework* de testes unitários

Seguidamente irão ser descritas as classes que compõem a *framework*:

- ***Template_Controller***: permite desenhar as vistas.
- ***Vendor_User_Controller***: exemplo de um controlador da aplicação.
- ***User_Model*** e ***Vendor_Model***: exemplos de modelos da aplicação.
- **ORM**: contém as funções de ORM do *Kohana* e possibilita a obtenção de objetos com informação de registos de uma base de dados relacional.
- ***TestSuite***: representa as baterias de testes do *SimpleTest* e permite a adição de testes para serem executados nas baterias.

- ***AllTests***: corresponde a uma bateria de todos os testes a executar. É uma classe que é preciso criar de raiz caso se pretenda agrupar os testes em baterias. No entanto é totalmente opcional e os testes podem ser executados individualmente, não se justificando nesse caso a existência desta classe.
- ***UnitTestCase***: constitui um caso de teste do *SimpleTest*.
- ***VendorUser_Test***: representa um caso de teste aos modelos ***User_Model*** e ***Vendor_Model*** da aplicação. É a única classe que é preciso criar de raiz para efetuar testes à aplicação.
- ***Mock***: pertence ao *SimpleTest* e permite gerar o código das classes *mock*.
- ***Phactory***: corresponde às funções do *Phactory* e possibilita estabelecer a conexão à base de dados, criar tabelas, inserir e obter dados das tabelas, entre outras funcionalidades.

Apresentada a estrutura da *framework* de testes unitários, dos componentes que a constituem e respetivas funções, irá ser abordada no próximo capítulo a criação do novo módulo de testes unitários.

3.12 Criação do Novo Módulo de Testes Unitários

A criação de um módulo de testes unitários foi necessária devido ao facto de ser preciso separar os ficheiros da aplicação (plataforma de encomendas *online* da *LabOrders*) e dos testes. Isto facilita a localização, edição e integração dos ficheiros na medida em que permite ter um repositório principal para a aplicação e um subrepositório para os testes.

Como tal foi criado o diretório *"C:\wamp\www\kohana_unit_tests\modules\unit_tests\"* dentro do qual se replicou parcialmente a estrutura de ficheiros do *Kohana*, com a criação das pastas *controllers* e *tests*. Desta forma os controladores da aplicação são colocados no diretório principal do *Kohana*: *"C:\wamp\www\kohana_unit_tests\application\"*, na pasta *controllers*, e as baterias de testes são colocadas na pasta *controllers* do módulo, bem como os ficheiros de teste na pasta *tests*.

A ativação do módulo é feita usando a definição '*modules*' adicionando o seguinte código ao ficheiro *"C:\wamp\www\kohana_unit_tests\application\config\config.php"*:

```
// Paths are relative to the docroot, but absolute paths are also possible
// Use the MODPATH constant (?)
$config['modules'] => array(MODPATH . 'unit_tests')
```

Código 15 – Ativação do novo módulo de testes unitários

De seguida foi definido o subrepositório dos testes (o repositório principal da aplicação já existia), de forma a separar os *commits* (registos das alterações efetuadas) dos testes e da aplicação. Para isso foi necessário:

1. Criar o repositório *unit_tests* no *Bitbucket*: https://bitbucket.org/.
    a. Instalar o *TortoiseHg*: http://bitbucket.org/tortoisehg/files/downloads/tortoisehg-3.5.0-x64.msi.
    b. Criar uma conta no *Bitbucket*.
    c. Clicar em ***Create*** -> ***Create repository***.
    d. Introduzir *unit_tests* como nome do repositório e clicar no botão ***Create repository***.
    e. Em ***Actions***, clicar na opção ***Clone*** e copiar a hiperligação, por exemplo https://gcastelo@bitbucket.org/gcastelo/teste.
    f. Clicar com o botão direito do rato na pasta do módulo, clicar em ***TortoiseHg*** e de seguida em ***Clone…***.
    g. Colar a hiperligação obtida no passo e. em **Origem…** e clicar no botão **Clonar**.

2. Criar um ficheiro .hignore na raíz do repositório principal, de forma a ignorar, ou seja, não fazer *commit* dos ficheiros do subrepositório para o repositório principal, com o seguinte conteúdo:

```
# use glob syntax
syntax: glob

modules/unit_tests/**
```

Código 16 – Ficheiro .hignore para não fazer *commit* dos ficheiros do subrepositório para o repositório principal da aplicação

3. Fazer *commit* e *push* (envio das alterações para o repositório remoto [*Bitbucket*]).

Neste ponto a *LabOrders* já se encontrava preparada para codificar todos os testes unitários que fossem necessários, no entanto foi sugerido o desenvolvimento de uma mini-aplicação como prova de conceito que foi aceite e será abordado no capítulo seguinte.

3.13 Desenvolvimento da Aplicação de Testes Unitários

A aplicação de testes desenvolvida como prova de conceito, cujas funcionalidades foram definidas em diversas reuniões com os orientadores, tiveram como principal objectivo um melhor entendimento do funcionamento dos *mock objects* e da *mock database* e que permite obter:

- Todos os utilizadores associados a um determinado fornecedor.
- Os nomes de todos os fornecedores.
- Fornecedores por *id*.
- O nome de um determinado utilizador.
- O apelido de um determinado utilizador.
- O fornecedor associado a um determinado utilizador.
- O número total de utilizadores.

O diagrama de sequência da **Figura 18** descreve o funcionamento da aplicação. O utilizador começa por invocar a função `list_all()` que desenha a vista. O controlador cria a nova vista e obtém um objeto do tipo *vendor* (fornecedor) utilizando o método `factory()` do ORM do *Kohana*. De seguida obtém os nomes dos fornecedores invocando a função `get_vendor_names()` do modelo do fornecedor e define o valor do atributo na vista. Repete-se o procedimento para a obtenção dos fornecedores mas desta vez invocando a função `find_all()` do ORM do *Kohana*. O controlador desenha a vista com o *método* `_render()`. É apresentado ao utilizador um formulário que contém uma caixa de lista suspensa (*dropdown list*) com os nomes de todos os fornecedores e um botão *OK*. O utilizador seleciona um fornecedor e clica no botão. Os dados são enviados via *POST* para a vista que é redesenhada da forma a apresentar todos os utilizadores associados ao fornecedor selecionado. Entende-se por *POST* o método de enviar dados através de um formulário em que os dados são incluídos no corpo da mensagem.

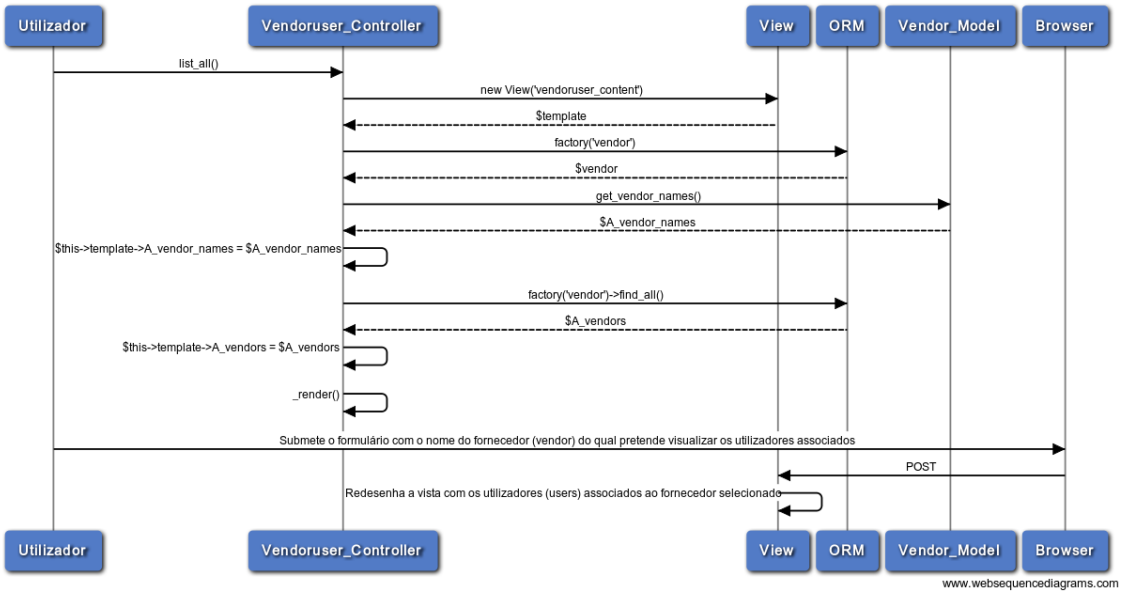

Figura 18 – Descrição da aplicação (prova de conceito)

Para aceder à vista (**Figura 19**) basta introduzir no *browser* o endereço: http://ip_da_maquina/nome_do_diretorio/index.php/nome_do_controlador/nome_da_funcao_que_desenha_a_vista, que neste caso é http://127.0.0.1/kohana_unit_tests/index.php/vendoruser/list_all porque está a ser executado no servidor local.

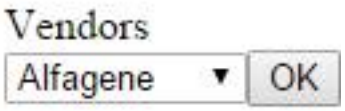

Figura 19 – Vista da aplicação de testes (parte 1)

Ao clicar no botão *OK* são apresentados todos os utilizadores associados ao fornecedor selecionado (**Figura 20**).

Alfagene users

id	vendor_id	name
1	1	Luis Guilherme
2	1	Gomes Albuquerque

Figura 20 – Vista da aplicação de testes (parte 2)

O código pode ser consultado no anexo **"Aplicação de Testes"**.

Após a conclusão da aplicação resta apenas testá-la. Para criar um *mock object* invoca-se o método `generate()` da classe *Mock* do *SimpleTest* que permite criar um modelo *mock* (simulação do modelo real).

```
// this class generates a class called 'Vendor_Model'
// with the same interface as 'Vendor_Model'
Mock::generate('Vendor_Model');
```

Código 17 – Exemplo de criação de um modelo *mock*

Agora é possível criar instâncias do modelo *mock* (*Vendor_Model*) no teste:

```
$vendor1 = new MockVendor_Model('vendor', 1);
```

Código 18 – Exemplo de instanciação de um *mock object*

Para forçar o valor de retorno de uma função a ser um vetor de utilizadores, por exemplo, deve-se proceder da seguinte forma [a função `get_users()` irá retornar o vetor de utilizadores definido]:

```
$mockedUsers1 = array(
    array('id' => 1, 'vendor_id' => 1, 'name' => 'Luis Guilherme'),
    array('id' => 2, 'vendor_id' => 1, 'name' => 'Gomes Albuquerque'),
);

$vendor1->returns('get_users', $mockedUsers1);
```

Código 19 – Exemplo da definição do valor de retorno de uma função

Os *mock objects* são úteis quando se depende de valores da base de dados para testar o comportamento da aplicação. Por exemplo:

```
public function test_square_users()
{
    $user_model = new MockUser_Model('user');

    $user_model->returns('get_number_users', 10);

    $n_users = $user_model->get_number_users();

    $square_users = pow($n_users, 2);

    $this->assertEqual($square_users, 100);
}
```

Código 20 – Exemplo de um teste com *mock objects*

O valor de retorno do método `get_number_users()` (dependência da base de dados) é necessário para realizar a asserção e se ter a certeza que o quadrado do número total de utilizadores está a ser calculado corretamente. Deve realizar-se o isolamento da dependência através dos *mock objects* (forçando o valor de retorno dos métodos) de forma a assegurar que qualquer eventual erro se deve ao código e não ao facto do servidor de base de dados estar em baixo.

Em relação à *mock database*, a sua finalidade é testar a integridade dos dados. A função `setUp()` é executada antes de cada teste e tem como objetivo estabelecer a ligação à *mock database*:

```
function setup() {
        // Database connection or queries here
        // create a db connection and tell Phactory to use it
        $pdo = new PDO('mysql:host=127.0.0.1; dbname=testdb', 'root', '');
        Phactory::setConnection($pdo);

        // reset any existing blueprints and empty any tables Phactory has used
        Phactory::reset();
}
```

Código 21 – Ligação à *mock database*

Por sua vez a função `tearDown()` é executada depois de cada teste com o intuito de limpar a informação que foi armazenada na *mock database* se for caso disso, ou seja, quando não se pretende reutilizar a informação do teste anterior:

```
function tearDown() {
    Phactory::reset();
}
```

Código 22 – Eliminação da informação armazenada na *mock database*

Para criar um registo na *mock database* é necessário chamar o método `create()` do *Phactory*. É retornado um objeto *Phactory_Row* com a mesma informação do registo criado na *mock database*. Este objeto será utilizado para verificar se os métodos da aplicação estão a obter os mesmos dados da *mock database* que foram criados (integridade dos dados):

```
// create a row in the db with id = 1 and name = 'Alfagene'
Phactory::create('vendor', array('id' => 1, 'name' => 'Alfagene'));

// create a row in the db with id = 1, vendor_id = 1 and name = 'Luis
Guilherme', and store a Phactory_Row object
       $user1 = Phactory::create('user', array('id' => 1, 'vendor_id' => 1,
'name' => 'Luis Guilherme'));

// get the vendor with id = 1 using the Kohana ORM
$vendor1 = ORM::factory('vendor', 1);

$A_users1 = $vendor1->get_users();

$vendor_id1 = $A_users1[0]->vendor_id;

$this->assertEqual($user1->vendor_id, $vendor_id1);
```

Código 23 – Exemplo de criação de um registo na *mock database*, obtenção dos seus dados e asserção sobre esse conteúdo

Quando as chaves primárias são auto incrementadas não é prático obter os objetos utilizando o ORM do *Kohana* porque não se sabe de uma forma imediata quais os ids que o teste criou. É preferível usar a função `get()` do *Phactory* com outra informação que não o *id*, por exemplo:

```
$joe = Phactory::get('user', array('name' => 'Joe Smith', 'age' => 20,
'activated' => 1));
```

Código 24 – Exemplo de obtenção de um objeto através do ORM do *Phactory*

Assim é obtido o utilizador cujo nome (*name*) é Joe Smith, cuja idade (*age*) é 20 e que se encontra ativo no sistema (1 = ativo, 0 = desativado).

O *Phactory* também fornece a possibilidade dos valores por defeito conterem uma sequência incrementada automaticamente. Utilizando esta funcionalidade, cada objeto sucessivo criado numa tabela irá ter um valor diferente mesmo que não se sobrecarreguem (*override*) os valores por defeito. Para usar uma sequência é preciso apenas definir um esquema (*blueprint*) da tabela através do método `define()` e passar-lhe $n como parâmetro, por exemplo:

```
Phactory::define('user', array('name' => 'user$n', 'age' => '$n'));

$user0 = Phactory::create('user'); // name == 'user0', age == '0'
$user1 = Phactory::create('user'); // name == 'user1', age == '1'
$user2 = Phactory::create('user'); // name == 'user2', age == '2'
```

Código 25 – Exemplo de criação de objetos utilizando sequências

O $n irá ser substituído por um número que começa em 0 e é incrementado por 1 cada vez que se cria um objeto na tabela. Neste caso temos definidos no esquema da tabela os campos '*name*' e '*age*' com os valores '*user$n*' e '*$n*', respetivamente. Sendo assim o primeiro utilizador irá ter como '*name*' '*user0*' e como '*age*' '0'. O segundo utilizador irá ter como '*name*' '*user1*', '*age*' '1' e por aí adiante…

O ORM do *Kohana* e do *Phactory* possibilita igualmente a definição de associações entre objetos.

Figura 21 – Associação entre fornecedores e utilizadores

Por exemplo, para definir a associação entre as tabelas *vendors* (fornecedores) e *users* (utilizadores) da **Figura 21** é preciso incluir no modelo do fornecedor:

```
protected $has_many = array('users');
```

E no modelo do utilizador:

```
protected $belongs_to = array('vendor');
```

Desta forma um fornecedor tem vários utilizadores e um utilizador pertence a um único fornecedor.

Inclui-se de seguida, como exemplo, o diagrama de sequência da **Figura 22** que descreve o teste da obtenção dos utilizadores associados a um determinado fornecedor, com *mock database*.

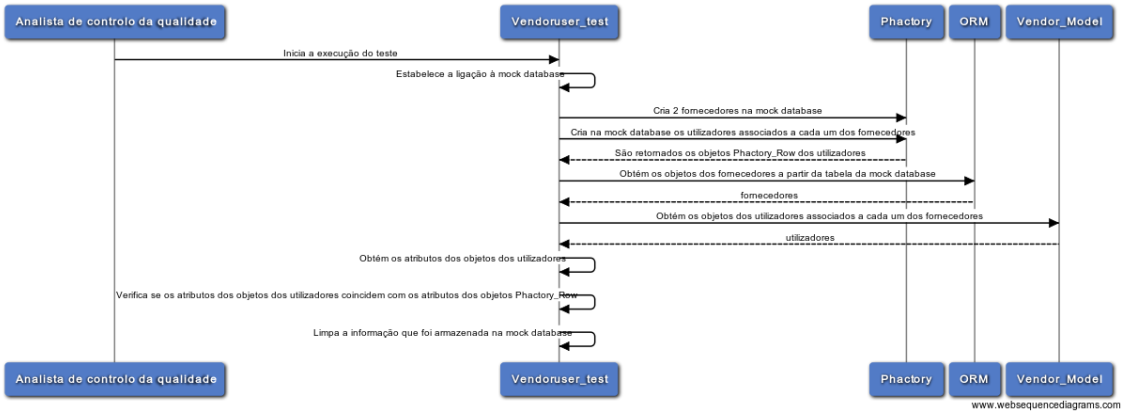

Figura 22 – Descrição do teste de obtenção de utilizadores

O código pode ser consultado no anexo **"Teste da Obtenção de Utilizadores"**.

Foram também codificados testes unitários para a obtenção do nome/apelido/fornecedor de um determinado utilizador.

Finalmente é possível agrupar vários casos de teste numa bateria de testes:

```
require_once('\libraries\simpletest\autorun.php');
require_once('\libraries\Phactory\lib\Phactory.php');

class AllTests extends TestSuite {
    function __construct() {
        parent::__construct();

        $this->addFile('\application\tests\test.php');

        $this->addFile('\application\tests\myTestCase.php');

        $this->addFile('\application\tests\phactory_test.php');
    }
}
```

Código 26 – Exemplo de uma bateria de testes

A função `addFile()` é utilizada para adicionar os casos de teste a serem executados na bateria de testes.

Para correr uma bateria de testes (**Figura 23**), basta escrever http://ip_da_maquina/kohana/index.php/nome_da_bateria_de_testes na barra de endereços do *browser*, por exemplo: http://127.0.0.1/kohana/index.php/all_tests (porque durante o estágio a execução foi feita no servidor local).

Figura 23 – Resultado dos testes

Efetuada a demonstração da aplicação de testes na qual se pôde observar que tudo se encontrava a funcionar corretamente, faltava somente integrar a *framework* de testes unitários desenvolvida com a plataforma de encomendas *online* da *LabOrders*.

No entanto, tal não foi possível devido ao facto da empresa não ter demonstrado abertura para fornecer o código e a estrutura da base de dados da aplicação para este efeito e, por conseguinte, não foi possível realizar a etapa referida. De qualquer das formas, o processo de configuração do projeto final é o mesmo da aplicação de testes desenvolvida como prova de conceito, que se encontra detalhado no anexo **"Configuração do *Kohana*, *SimpleTest* e *Phactory*"**. Para além disso também foi documentado no *Redmine* da *LabOrders*, logo a empresa reúne todas as condições para integrar a *framework* de testes unitários com qualquer *software*, caso assim o entenda.

4 Avaliação

A metodologia de avaliação seguida baseou-se nos valores fundamentais e nas boas práticas da XP. O processo de garantia da qualidade foi efetuado através dos testes automáticos que asseguram que todo o trabalho desenvolvido se encontra a funcionar corretamente, sendo esse um dos objetivos do projeto de mestrado. O código foi sempre revisto por grupos de duas pessoas (o autor da tese e um funcionário da *LabOrders*) para detetar mais facilmente erros de lógica e sintaxe, no sentido de tornar o *software* mais eficaz, eficiente e intuitivo. Sempre que os revisores de código chegavam à conclusão que era possível simplificá-lo sem afetar nenhuma funcionalidade procedia-se à sua refatorização. As revisões eram constantes de forma a permitir a integração contínua dos novos testes, funcionalidades e correções com o código dos outros programadores. A bateria de testes era então executada nos computadores dos colaboradores da empresa, que realizavam a sua avaliação dos resultados e forneciam o seu *feedback*. Isto possibilita a deteção imediata e respetiva resolução de falhas relacionadas com configurações erradas, falta de permissões, conflitos entre versões do código ou incompatibilidades entre o *software* instalado num posto de trabalho em específico. Em princípio nesta fase já não devem ocorrer erros de código uma vez que os testes foram executados previamente com sucesso no computador da pessoa que os codificou. Todo o código desenvolvido foi aprovado pela *LabOrders* o que se traduz num *feedback* 100% positivo por parte da empresa. O fluxograma da **Figura 24** descreve o método de avaliação do código.

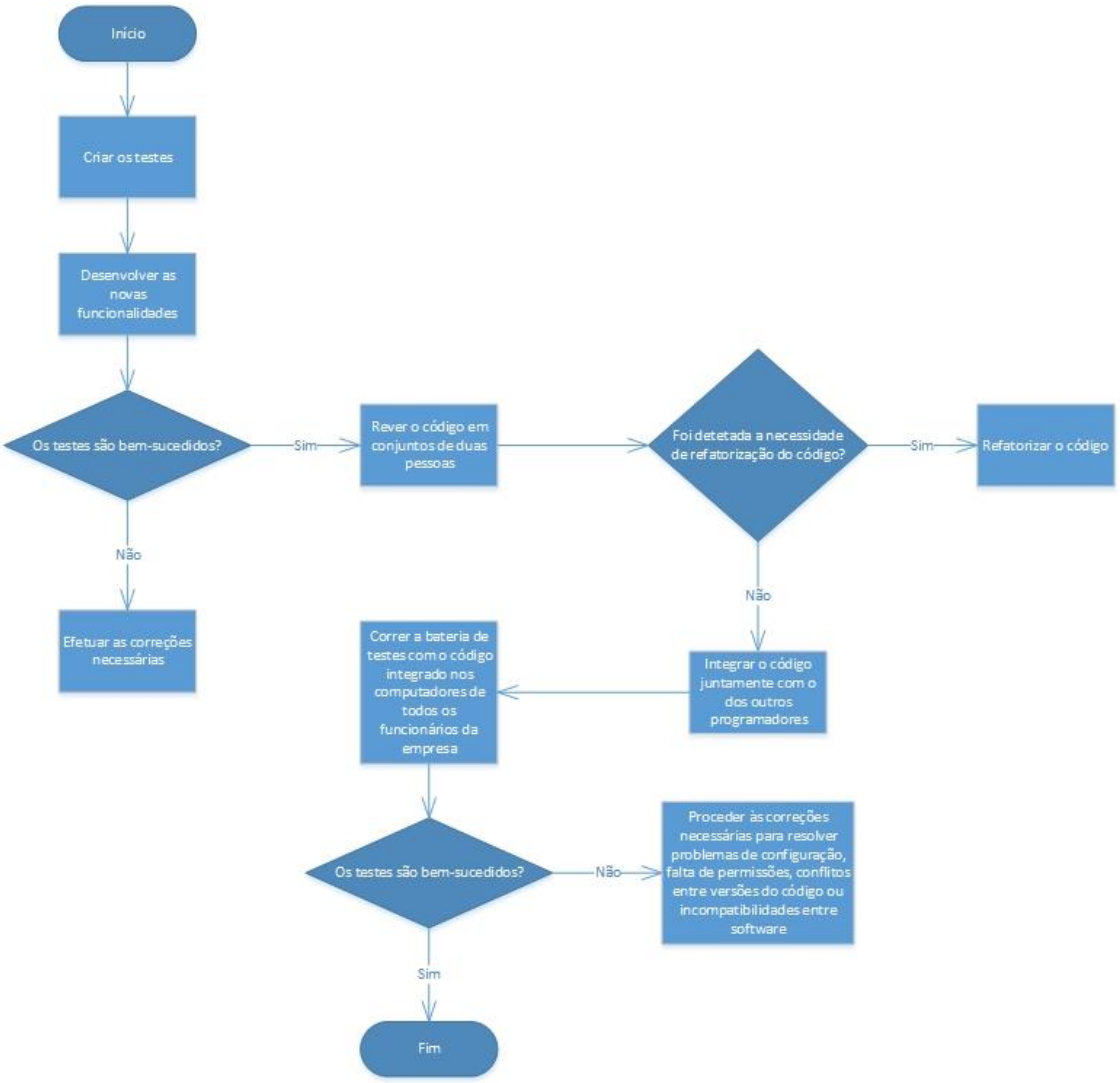

Figura 24 – Processo de garantia da qualidade/método de avaliação do código

Os clientes também requisitam a implementação de novas funcionalidades na plataforma *online* de encomendas, para além das previstas pela *LabOrders*. Isto resulta, consequentemente, na criação de mais testes.

Em relação aos *sites WordPress*, o desenvolvimento foi feito diretamente no servidor remoto com cópias de segurança (*backups*) dos ficheiros e base de dados para o servidor local, o que possibilita que os clientes deêm a sua opinião e indiquem alterações em tempo útil.

Por fim, a importação de catálogos de produtos para a plataforma de encomendas era feita imediatamente após a geração dos ficheiros, depois do seu conteúdo ser analisado e aprovado pelo Supervisor Externo, o Doutor Tiago Carvalho. Nos raros casos em que o cliente pretendia comprar um produto que não se encontrava disponível, comunicava com o fornecedor que por sua vez entrava em contacto com a *LabOrders* para o produto ser adicionado. Era mantido um fluxo de comunicação constante entre todos: líderes, membros das equipas e utilizadores.

No entanto o contacto com os clientes era sempre feito por intermédio do Doutor Tiago Carvalho, Diretor Geral da *LabOrders*. Por conseguinte a avaliação do trabalho junto dos clientes foi feita por si. Transmitiu ao autor da tese que os clientes se mostraram totalmente satisfeitos com as correções das falhas detetadas pelos testes, as importações de catálogos e os *sites WordPress* desenvolvidos, o que se traduz num *feedback* 100% positivo por parte dos clientes.

Outro processo de avaliação passou por inquirir oralmente sobre o trabalho todos os funcionários da empresa com que se contactou diretamente (3 funcionários), o que permitiu a obtenção de dados estatísticos que refletem o seu grau de satisfação com o mesmo. Apresentam-se de seguida as questões colocadas e as respetivas respostas:

Q1 – A reconfiguração total e correta do *Selenium WebDriver* foi um aspeto positivo do trabalho desenvolvido?

R1 – Sim, claramente. Antes da reconfiguração apenas era possível executar os testes numa versão desatualizada do *Firefox* e após a reconfiguração passou a ser possível executar os testes nas versões mais recentes de todos os *browsers* que a *LabOrders* utiliza: *Firefox*, *Chrome* e *Internet Explorer*.

(Todos os funcionários [100%] consideram a reconfiguração total e correcta do *Selenium WebDriver* como um aspeto positivo do trabalho desenvolvido).

Q2 – Qual a sua opinião sobre a exibição de mensagens de erro personalizadas?

R2 – As mensagens de erro personalizadas são úteis porque se não forem definidas, quando o *Selenium WebDriver* não encontra um determinado elemento retorna uma mensagem de erro muito pouco explicativa: "*Element not found*", o que dificulta imenso a identificação do elemento que origina o erro. Definindo mensagens de erro personalizadas estas são exibidas sempre que o *Selenium WebDriver* não consegue encontrar um elemento, permitindo identificar de imediato esse elemento e proceder às correções necessárias. Desta forma a codificação dos testes torna-se mais rápida, simples e intuitiva.

(Todos os funcionários [100%] consideram que a exibição de mensagens de erro personalizadas é útil).

Q3 – E sobre a análise do conteúdo de ficheiros .pdf?

R3 – É essencial na medida em que permite verificar se a informação das ordens de encomenda e dos avisos de receção gerados pela plataforma de encomendas *online* da *LabOrders* contêm os nomes dos fornecedores, os nomes dos produtos e os preços corretos.

(Todos os funcionários [100%] consideram que a análise do conteúdo de ficheiros .pdf é uma funcionalidade essencial).

Q4 – E sobre a obtenção de elementos *web* através de código *jQuery*?

R4 – A funcionalidade de pesquisa invertida do *jQuery* traz vantagens em situações em que se precisa de obter elementos que contêm outros elementos, por exemplo um formulário que contém um determinado botão. Se o formulário não tiver atributos suficientes para que o *Selenium WebDriver* o possa obter a partir dos seus seletores, através de código *jQuery* é possível obter primeiro o botão e realizando a pesquisa invertida obter de seguida o formulário que o contém.

(Todos os funcionários [100%] consideram que a obtenção de elementos *web* através de código *jQuery* traz vantagens).

Q5 – E sobre o *download* automático de ficheiros através do *Internet Explorer*?

R5 – É importante porque antes desta funcionalidade ter sido implementada todos os testes que envolviam o *download* de ficheiros falhavam no *Internet Explorer* em circunstâncias nas quais não deviam falhar uma vez que tinham sucesso nos outros dois *browsers* que a empresa utiliza, o *Firefox* e o *Chrome*. Após a criação desta funcionalidade os testes em questão passaram a ser bem-sucedidos. Só falham quando existe efetivamente algum problema com o *download* do ficheiro, ou seja, quando devem mesmo falhar.

(Todos os funcionários [100%] consideram que o *download* automático de ficheiros através do *Internet Explorer* é importante).

Q6 – A criação da *framework* de testes unitários com dependências trouxe benefícios?

R6 – Sem dúvida. Permite testar o comportamento da aplicação (plataforma de encomendas) simulando as respostas dos serviços através dos *mock objects*, e também testar a integridade dos dados através da *mock database*. Sem a *framework* de testes unitários com dependências nada disto seria possível.

(Todos os funcionários [100%] consideram que a criação da *framework* de testes unitários com dependências trouxe benefícios).

Q7 – E a criação do novo módulo de testes unitários?

R7 – Foi bastante vantajosa porque permite ter os controladores da aplicação e as baterias de testes em diretórios separados. Se estivessem os ficheiros todos no mesmo diretório tornava-se muito confuso, desta forma a localização de ficheiros torna-se mais fácil. Para além disso possibilita ter um repositório principal para a aplicação e um subrepositório para os testes separando os *commits* de cada um, caso contrário seria complicado distinguir as alterações efetuadas à aplicação das alterações efetuadas aos testes.

(Todos os funcionários [100%] consideram que a criação do novo módulo de testes unitários foi vantajosa).

Q8 – O código passou a ter maior qualidade com a adoção da XP?

R8 – Sim, evidentemente. O TFD implica que os testes sejam escritos antes das funcionalidades. Requer também que a pessoa que criou o teste seja a mesma que vai implementar a funcionalidade respetiva. Desta forma os elementos web terão todos os atributos necessários à realização dos testes, uma vez que o programador sabe de antemão a estratégia de localização que definiu no teste e os elementos que precisa de obter.

Por sua vez a refatorização torna o código mais simples, modularizado e legível. Torna-o igualmente mais fácil de compreender, modificar e manter.

(Todos os funcionários [100%] consideram que o código passou a ter mais qualidade com a adoção da XP).

Q9 – Os testes são uma mais-valia? Aumentaram a produtividade da empresa? Vieram melhorar o trabalho?

R9 – Os testes automáticos são obviamente uma mais-valia. Reduzem o erro humano e podem ser executados nas diferentes versões da aplicação (alfa, desenvolvimento, produção, entre outras) dando uma boa cobertura de testes. Isto resulta em mais testes executados em menos tempo e utilizando menos recursos (do que se fossem realizados manualmente), o que permite reduzir custos e aumentar a produtividade da empresa. São detetados erros que antes passavam em claro e resultavam em diversos problemas cuja origem não se conseguia identificar, perdendo-se imenso tempo desnecessariamente nesse processo de inspecção. Os testes permitem resolver rapidamente os erros e libertam os funcionários para a execução de outras tarefas mais interessantes como o desenvolvimento de novas funcionalidades, melhorando o seu trabalho.

(Todos os funcionários [100%] consideram que os testes são uma mais-valia, aumentaram a produtividade da empresa e vieram melhorar o trabalho).

Não foi possível inquirir elementos externos à *LabOrders* devido à política de privacidade da empresa.

Os resultados deste estudo permitem concluir que a opinião dos clientes, dos funcionários e da *LabOrders* em geral sobre o trabalho desenvolvido é 100% positiva, o que condiz com o parecer escrito emitido pela empresa no final do estágio, também ele totalmente positivo.

5 Conclusão

A escolha de uma metodologia de desenvolvimento de *software* adequada é essencial para o aumento da produtividade e redução de falhas, bem como o cumprimento de orçamentos e prazos dos projetos das organizações.

O desenvolvimento evolucionário possui inúmeras vantagens em relação ao desenvolvimento sequencial. As metodologias ágeis refletem maiores probabilidades de sucesso e permitem a criação rápida de *software* com mais qualidade e menos erros do que as metodologias tradicionais. Isto explica o facto de se encontrarem tão em voga e terem ultrapassado as metodologias pesadas em número de utilizadores. Nos últimos 20 anos as metodologias ágeis não pararam de evoluir, ao passo que as metodologias orientadas a documentação têm regredido, existindo hoje em dia mais projetos ágeis do que em cascata [Standish Group, 1995] [Version One, 2013].

A tese realizada permitiu responder à questão "Será que realmente compensa realizar testes (automáticos)?" com base nos dados estatísticos resultantes do trabalho desenvolvido e que já foram apresentados no capítulo **"Avaliação"**. Os dados estatísticos obtidos a partir dos estudos abordados no subcapítulo **"Vantagens das Metodologias Ágeis Face às Metologias Tradicionais"** do capítulo **"Metodologias de Desenvolvimento e Testes Automáticos de *Software*"** (estado da arte) também contribuem para a fundamentação da resposta. Realmente compensa realizar testes automáticos na medida em que, juntamente com a refatorização, são cruciais no processo de controlo da qualidade do *software* promovendo a criação de uma especificação detalhada e uma simplificação radical do código com um *design* limpo e intuitivo [Karai, 2009]. O erro humano é reduzido, as empresas poupam tempo e dinheiro, é dada uma boa cobertura de testes e fornecida uma vasta gama de tolerância para a verificação dos requisitos não funcionais (parâmetros subjetivos).

As ferramentas de testes automáticos ainda se encontram num patamar evolutivo abaixo das ferramentas de codificação de *software*, muitas áreas não são abrangidas pelos testes e por isso alguns têm de ser feitos manualmente [Haria]. No entanto os avanços nesta área têm sido notórios [Standish Group, 1995] [Version One, 2013] e o trabalho efetuado pretende dar continuidade a esse progresso, promovendo a criação de soluções mais completas, incentivando a utilização das metodologias ágeis de desenvolvimento de *software* e porventura inspirando a realização de outros projetos sobre a temática.

5.1 Objetivos Realizados

De forma a dar sequência à evolução dos testes automáticos de *software* começou por ser configurada a *framework* de testes funcionais e unitários sem dependências, com o objetivo de garantir o seu funcionamento nos três *browsers* que a *LabOrders* utiliza: *Firefox*, *Chrome* e *Internet Explorer*. De seguida foram desenvolvidas várias funcionalidades inovadoras que não se encontram disponíveis nas ferramentas analisadas no âmbito deste documento. Essas funcionalidades são:

- O *download* automático de ficheiros através do *Internet Explorer 9*.
- A exibição de mensagens de erro personalizadas.
- A análise do conteúdo de ficheiros .pdf.
- A obtenção de elementos *web* e respetivos atributos através de código *jQuery* (numa *framework* que se destina a que os testes sejam escritos maioritariamente em PHP e não em *JavaScript*).

Adicionalmente, foram corrigidos vários testes codificados pelos colaboradores da empresa antes do projeto de mestrado ter tido início, assim como a funcionalidade de envio do relatório de testes por correio eletrónico. Numa fase posterior foi também requisitada a realização de um segundo projeto não previsto inicialmente, que consistiu no desenvolvimento de uma *framework* de testes unitários com dependências externas (base de dados e serviços), e incluiu a implementação de uma aplicação de testes como prova de conceito, bem como a criação de um novo módulo de testes unitários.

Por sua vez a dissertação fornece uma visão das diferentes metodologias existentes, demonstra o progresso da tecnologia e fomenta a adoção do modelo evolucionário de desenvolvimento de *software* (metodologias ágeis), bem como de boas práticas de programação.

Em termos de trabalho suplementar foram codificadas várias dezenas de testes funcionais e unitários, realizadas 22 importações de catálogos de produtos para a plataforma de encomendas *online* da *LabOrders* e criados 3 *sites WordPress*. Apenas foram colocados nos anexos deste documento um exemplo de uma importação de catálogo e a página principal de um dos *sites* desenvolvidos. As restantes importações de produtos e páginas dos *sites* não foram incluídas devido ao facto do número de páginas ser limitado. De qualquer das formas todos os *sites* estão disponíveis para consulta *online* e a importação exemplificada abrange quase um milhão de produtos demonstrando uma das importações de catálogos efetuadas mais complexas.

Posto isto todos os objetivos propostos não só foram cumpridos como também foram ultrapassados. A tese tem o grau de complexidade e inovação exigido, tanto em termos qualitativos como quantitativos, visto que para além do projeto agendado no início foi ainda levado a cabo um segundo projeto, acrescido de uma grande quantidade de trabalho suplementar.

5.2 Limitações e Trabalho Futuro

A principal limitação foi a não integração da *framework* de testes unitários com a plataforma de encomendas, que pode ser efetuada como trabalho futuro caso a *LabOrders* assim o entenda, e deveu-se ao facto da empresa não se ter mostrado disponível para fornecer a estrutura da base de dados e o código necessários para o efeito. No entanto o processo de configuração é exatamente igual ao realizado na aplicação de testes, que foi documentado no *Redmine* da *LabOrders* e neste documento.

A transferência automática de ficheiros através do *Internet Explorer 9* (e versões mais recentes) é conseguida através de um *script AutoIT* e do gestor de *downloads Orbit* porque o *Selenium WebDriver* não permite manipular *IEFrames*. Como a *LabOrders* prefere não utilizar o *Orbit* poderá futuramente tomar a decisão lógica de usar o *Edge* (anteriormente conhecido como *Spartan*), o novo *browser* da *Microsoft*. Trata-se da evolução do *Internet Explorer* e o seu comportamento é semelhante ao do *Chrome* e *Firefox*, ou seja, não utiliza *IEFrames*/barras de notificação em que o comportamento padrão é abrir o ficheiro. Em vez disso inicia automaticamente o *download* para o diretório definido pelo utilizador.

Os testes incompletos não estão a ser exibidos como uma barra amarela na GUI do *NetBeans* devido ao facto do *PHPUnit* não os incluir no seu relatório XML. O erro já foi reportado e o problema será solucionado automaticamente a partir do momento em que a equipa do *PHPUnit* proceda às correções necessárias e o atualize. De qualquer das formas, esta limitação é ultrapassada devido ao facto de no *output* da consola os testes incompletos serem corretamente sinalizados com um I (de *Incomplete*).

As *frameworks* foram desenvolvidas tendo em consideração as necessidades futuras da *LabOrders* e estão preparadas para a criação de outros tipos de testes (de carga, integração, regressão, entre outros) conforme seja preciso. Estes testes poderão ser feitos da seguinte forma:

- Testes de carga: efetuar múltiplos pedidos ao servidor e verificar se a aplicação continua operacional (por exemplo se é possível obter um elemento da página principal).
- Testes de integração: Combinar os componentes da aplicação em grupos de forma a testar as suas interfaces. Se os resultados forem os esperados então a comunicação entre módulos foi bem-sucedida o que significa que todos os módulos do grupo se encontram a funcionar corretamente.
- Testes de regressão: Integrar o novo código com o código antigo da aplicação de forma a verificar se esta regrediu, ou seja, se ao juntar o novo código surgiram erros nos testes do código antigo que anteriormente não existiam.

Desta forma considera-se que a tese foi concluída com sucesso, o que condiz com o parecer totalmente positivo da empresa.

Referências

[Agile Alliance]	O que é a Aliança?, http://www.agilealliance.org/pt/ [último acesso: Nov 2014]
[Agile Alliance b]	Tdd, http://guide.agilealliance.org/guide/tdd.html [último acesso: Nov 2014]
[Agile Manifesto, 2004]	Agile Manifesto, http://agilemanifesto.org/ [último acesso: Nov 2014]
[Ambler]	Ambler, Scott W. Feature Driven Development (FDD) and Agile Modeling, http://www.agilemodeling.com/essays/fdd.htm [último acesso: Dez 2014]
[Bastos, 2013]	Bastos, L. Quais vantagens do desenvolvimento ágil de software?, http://computerworld.com.br/blog/opiniao/2013/10/21/quais-vantagens-do-desenvolvimento-agil-de-software/, 21 Out 2013. [último acesso: Nov 2014]
[Beck, 1999]	Beck, K. Programação Extrema Explicada, Bookman, 1999.
[Beck, 2002]	Beck, K. Test Driven Development: By Example, Addison-Wesley, 1ª Edição, 18 Nov 2002.
[Beck, 2007]	Beck, K. Test-Driven Development Violates the Dichotomies of Testing, Three Rivers Institute, http://www.threeriversinstitute.org/Testing%20Dichotomies%20and%20TDD.htm, Jun 2007. [último acesso: Jan 2015]
[Belem, 2013]	Belem, T. Mas afinal, o que é o MVC? http://blog.thiagobelem.net/o-que-e-o-mvc/, 23 Jan 2013. [último acesso: Out 2015]
[Brooks, 1987]	Brooks, F. No Silver Bullet: Essence and Accidents of Software Engineering, Proc. IFIP, IEEE CS Press, pp. 1069-1076; reprinted in IEEE Computer, pp. 10-19, Abr 1987.
[Bukowicz, 2012]	Bukowicz, M. Integrating jQuery With Selenium, http://www.code-thrill.com/2012/04/integrating-jquery-with-selenium.html, 2 Abr 2012. [último acesso: Mar 2015]
[Cockburn, 2009]	Cockburn, A. Crystal methodologies, http://alistair.cockburn.us/Crystal+methodologies, 2009. [último acesso: Dez 2014]
[Cockburn, 2011]	Cockburn, A. Crystal Clear Methodology, http://c2.com/cgi/wiki?CrystalClearMethodology, 5 Set, 2011. [último acesso: Dez 2014]
[Cohn, 2007]	Cohn, M. Differences Between Scrum and Extreme Programming, http://www.mountaingoatsoftware.com/blog/differences-between-scrum-and-extreme-programming, 6 Abr 2007. [último acesso: Dez 2014]
[Cygnet Infotech, 2013]	Selenium vs QTP vs TestingWhiz, http://www.cygnet-infotech.com/blog/selenium-vs-qtp-vs-testingwhiz, 17 Jun, 2013. [último acesso: Out 2015]
[Dustin, 2001]	Dustin, E. The Automated Testing Lifecycle Methodology (ATLM), http://www.informit.com/articles/article.aspx?p=21468&seqNum=5, 25 Mai 2001. [último acesso: Jan 2015]
[Galin, 2003]	Galin, D. Software Quality Assurance: From Theory to Implementation, pp. 235-245
[Gilb, 1988]	Gilb, T. Principles of Software Engineering Management, Addison-Wesley, 1988.
[Haria]	Haria, M. Report On Automated Software Testing, http://www.cs.nott.ac.uk/~cah/G53QAT/Reports09/mxh06u/QAT09Report-mxh06u.doc [último acesso: Jan 2015]
[Harper, 2012]	Harper, J. Why the Name “Scrum”?, https://agilecoachjacque.wordpress.com/2012/12/04/why-the-name-scrum/, 4 Dez 2012. [último acesso: Dez 2014]

[Highsmith et al., 2000] Highsmith, J. Orr, K. Cockburn, A. Extreme Programming. E-Business Application Delivery, pp. 4-17, Fev 2000.

[James, 2009] James, M. Scrum Methodology, http://scrummethodology.com/, 2009. [último acesso: Dez 2014]

[Karai, 2009] Karai, V. Test Driven Development In An Agile Environment, http://pt.slideshare.net/vkarai/agile-test-driven-development-1265878, 19 Jan 2009. [último acesso: Jan 2015]

[Kite, 2011] Kite, C. Phactory A Database Factory for PHP, http://pt.slideshare.net/chriskite/phactory, 4 Fev 2011. [último acesso: Jun 2015]

[Kohana User Guide] Inflector, https://kohanaframework.org/3.0/guide/api/Inflector [último acesso: Out 2015]

[Maeda, 2002] Maeda, S. The Ruby Language FAQ, http://ruby-doc.org/docs/ruby-doc-bundle/FAQ/FAQ.html, 17 Dez 2002. [último acesso: Fev 2015]

[Maruvada, 2014 Maruvada, N. TDD – A paradigm shift within the software development process for the Agile Environment, http://www.agilerecord.com/tdd-paradigm-shift-within-software-development-process-agile-environment/, 10 Jun 2014. [último acesso: Jan 2015]

[Mauch, Bassuday et al, 2012] Mauch, M. B., Bassuday, K. et al, Crystal methodologies, http://pt.slideshare.net/bassuday/crystal-methodology, 6 Mai 2012. [último acesso: Dez 2014]

[Millikin, 2014] Millikin, A. What Types of Software Testing Can and Should be Automated? http://www.seguetech.com/blog/2014/02/07/types-software-testing-can-should-automated, 7 Fev 2014. [último acesso: Jan 2015]

[Murphy, 1996] Murphy, K. So why did they decide to call it Java?, http://www.javaworld.com/article/2077265/core-java/so-why-did-they-decide-to-call-it-java-.html, 4 Out 1996. [último acesso: Fev 2015]

[nodebr, 2013] O que é a NPM do Node.JS, http://nodebr.com/o-que-e-a-npm-do-nodejs/, 19 Jul 2013. [último acesso: Fev 2015]

[Patton, 2005] Patton, R. Software Testing, Sams Publishing, 5 Ago 2005.

[Royce, 1970] Royce, W. W. Managing the development of large software systems: concepts and techniques, Proc. IEEE Westcon, Los Angeles, CA.

[Schwaber e Beedle, 2002] Schwaber, K. e Beedle, M. Agile Software Development with Scrum, NJ, Prentice-Hall, 2002.

[Semedo, 2012] Semedo, M. J. Ganhos de produtividade e sucesso de Metodologias Ágeis VS Metodologias em Cascata no desenvolvimento de projectos de software, http://recil.grupolusofona.pt/bitstream/handle/10437/6174/Disserta%C3%A7%C3%A3o-Maria%20Semedo%5Bentrega%5D.pdf?sequence=1, Mar 2012. [último acesso: Out 2015]

[Soares, 2004] Soares, M. S. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software, http://xps-project.googlecode.com/svn/!svn/bc/6/trunk/outros/Met._Ageis.pdf, 2004. [último acesso: Nov 2014]

[Sommervill e, 1995] Sommerville, I. Software engineering, Addison-Wesley, 5ª edição, 1995.

[Sousa, 2009] Sousa, S. Software Application Testing, http://www.my-project-management-expert.com/software-application-testing.html [último acesso: Jan 2015]

[Sousa, 2013] Sousa, L. Test Driven Development, an agile methodology, http://www.mindsource.pt/en/content/test-driven-development-agile-methodology [último acesso: Jan 2015]

[Stack Overflow, 2009] What is stdClass in PHP?, http://stackoverflow.com/questions/931407/what-is-stdclass-in-php, 31 Mai 2009. [último acesso: Out 2015]

[Standish Group, 1995] CHAOS report, 586 Olde Kings Highway. Dennis, MA 02638, USA, 1995.

[Sturlese, 2013] Sturlese, L. Internet Explorer: Disable “Do you want to open or save this file?” download prompt, http://9to5it.com/internet-explorer-disable-do-you-want-to-open-or-save-this-file-prompt/, 10 Jul 2013. [último acesso: Mar 2015]

[Techopedia] What does Debugging mean?, https://www.techopedia.com/definition/16373/debugging [último acesso: Out 2015]

[Test Driven Websites, 2010] Different types of automated tests, http://testdrivenwebsites.com/2010/06/20/different-types-of-automated-tests/, 20 Jun 2010. [último acesso: Jan 2015]

[VersionOne, 2013] The 9th Annual State of Agile Report, http://info.versionone.com/state-of-agile-development-survey-ninth.html, 2013. [último acesso: Jan 2015]

[Williams, 2006] Williams, L. White-Box Testing, pp. 60-61, 69, http://www.chaudhary.org/WhiteBox.pdf [último acesso: Jan 2015]

[Wilson, 2011] Wilson, A. Why CSS Locators are the way to go vs XPath, http://sauceio.com/index.php/tag/css-selectors/, 17 Mai 2011. [último acesso: Out 2015]

Anexos

Configuração do *Selenium WebDriver*, *PHPUnit*, *NetBeans* e WAMP

A *LabOrders* já estava a utilizar o *Selenium WebDriver* com PHP *bindings* para codificar e executar os seus testes funcionais quando o estágio teve início, conforme foi referido previamente no subcapítulo **"Ferramentas de Testes Funcionais"** do capítulo **"Metodologias de Desenvolvimento e Testes Automáticos de *Software*"** (estado da arte), e no subcapítulo **"Funcionalidades Desenvolvidas"** da **"Descrição Técnica"**. No entanto os testes apenas corriam na versão 19 do *Firefox* (muito desatualizada, de 19 de Fevereiro de 2013, a versão atual no momento da escrita deste documento é a 40, de 3 de Julho de 2015) e não corriam de todo no *Chrome* nem no *Internet Explorer*. Este era o problema de maior urgência e por conseguinte foi o primeiro a ser resolvido. O motivo da anomalia foi o facto do *Selenium WebDriver* ter sido incorretamente configurado pela pessoa responsável na altura e como tal foi necessária a reinstalação, reconfiguração e atualização da *framework*. Para esse efeito foi efetuado o *download* da:

- Versão 2.44 do *Selenium Server* (necessário para correr os testes):

http://selenium-release.storage.googleapis.com/2.44/selenium-server-standalone-2.44.0.jar.

- Versão 2.9 do *Chrome Driver* (necessário para correr os testes no *Chrome*): http://chromedriver.storage.googleapis.com/2.9/chromedriver_win32.zip.

- Versão 2.44.0 do *IEDriver* (necessário para correr os testes no *Internet Explorer*): http://selenium-release.storage.googleapis.com/2.44/IEDriverServer_Win32_2.44.0.zip.

O *Firefox Driver* já se encontra incutido no *Selenium* mas é preciso ter em consideração a compatibilidade com a versão do *browser*, como tal verificou-se no registo de alterações: http://selenium.googlecode.com/git/rb/CHANGES quais as versões compatíveis: "2.44.0 -> *Firefox 24*, *31*, *32* e *33*". Posto isto foi realizado o *download* da:

- Versão 33 (a versão mais recente compatível) do *Firefox*: https://ftp.mozilla.org/pub/mozilla.org/firefox/releases/33.0/win32/en-GB/Firefox%20Setup%2033.0.exe.

- Versão 39.0.2171.71 m do *Chrome*: http://www.google.com/chrome/.

- Versão 9.0.8112.16421 do *Internet Explorer*: http://www.microsoft.com/pt-pt/download/internet-explorer-9-details.aspx~.

Os três *browsers* foram instalados e foi criado um diretório: "*C:\selenium*" onde foram colocados: o *Selenium Server*, o *Chrome Driver* e o *IEDriver*. Para ser possível iniciar o *Selenium Server* de uma forma simples (através de duplo clique) foi criado um ficheiro *batch* (extensão .bat, ficheiro de texto que contém uma sequência de comandos para um sistema operativo) que foi colocado no ambiente de trabalho. O conteúdo desse ficheiro é o seguinte:

```
java –jar C:\selenium\selenium-server-standalone-2.44.0.jar –
Dwebdriver.firefox.profile=testes –
Dwebdriver.chrome.driver=C:\selenium\chromedriver.exe –
Dwebdriver.ie.driver=C:\selenium\IEDriverServer.exe
```

Código 27 – Ficheiro batch para iniciar o *Selenium Server*

O ficheiro invoca o *Selenium Server* e inclui os *drivers* dos diferentes *browsers* sendo que para o *Firefox* especifica o nome do perfil a ser utilizado. Neste caso tem o nome "testes" que é um perfil especialmente criado de forma a ser possível limpar os *cookies* no início e no final de cada teste, como se poderá verificar no anexo **"Configuração dos *Browsers*"**. Caso se quisesse utilizar o perfil por defeito o nome deveria ser "*default*".

De seguida foi feito o *download* da versão 0.91 dos PHP *bindings*: https://code.google.com/p/php-webdriver-bindings/downloads/detail?name=php-webdriver-bindings-0.9.1.zip cujo conteúdo foi importado para o projeto fornecido pela empresa que continha os testes codificados previamente. Os ficheiros da versão desatualizada dos PHP *bindings* foram substituídos pelos da versão atual mas não sem antes se acrescentar às novas classes as funções não nativas desenvolvidas pela empresa [espera por pedidos AJAX e mensagens de erro personalizadas no método `findElementBy()`] existentes nas classes antigas.

Como já foi mencionado no subcapítulo **"Ferramentas de Testes Funcionais"** do "Estado da Arte", a *LabOrders* utiliza o *PHPUnit* para fazer asserções em alguns dos seus testes funcionais codificados antes do estágio ter tido início e criar eventuais testes unitários sem dependências externas. Por isso foi necessário instalá-lo através do *PHP Extension and Application Repository* (PEAR), uma *framework* e sistema de distribuição para componentes PHP reutilizáveis. Para instalar o PEAR é preciso:

1. Instalar o WAMP, um ambiente de desenvolvimento *web* utilizado pela empresa: http://www.wampserver.com/en/.

2. Efetuar o *download* do PEAR: http://pear.php.net/go-pear.phar.

3. Criar o diretório "*C:\pear*" e colocar lá o ficheiro go-pear.phar.

4. Clicar no botão **Iniciar** do *Windows* e selecionar **Executar**.

5. Na caixa de diálogo Executar, escrever a seguinte sequência de comandos:

```
cd C:\pear
php go-pear.phar
```

Código 28 – Inicialização da instalação do PEAR

6. Selecionar o diretório de instalação: "*C:\pear*".

7. Fazer duplo clique no ficheiro: "*C:\pear\pear.bat*".

8. Fazer duplo clique no ficheiro: "*C:\pear\PEAR_ENV.reg*".

Com o PEAR disponível reúnem-se todas as condições para instalar a versão mais recente do *PHPUnit*. Para isso é necessário:

1. Clicar no botão **Iniciar** do *Windows* e selecionar **Executar**.

2. Na caixa de diálogo Executar, escrever a seguinte sequência de comandos:

```
pear-config-set auto_discover 1
pear upgrade
pear clear-cache
pear channel-update –f pear.php.net
pear channel-discover –a –f phpunit/PHPUnit
```

Código 29 – Comandos PEAR para a instalação do *PHPUnit*

Seguidamente procedeu-se à configuração do *NetBeans*:

1. Clicar em ***Tools*** -> ***Options*** -> **PHP**.

2. Na aba ***General***, na secção ***Global Include Path*** clicar em ***Add Folder . . .*** e selecionar o diretório "*C:\pear\pear\PHPUnit*".

3. Na aba ***Frameworks & Tools***, selecionar ***PHPUnit*** do lado esquerdo e na secção ***PHPUnit Script:*** selecionar o diretório: "*C:\pear\phpunit.bat*".

Para dar sequência à configuração foi realizado o *download* do Selenium Module for PHP:

1. Clicar em ***Tools*** -> ***Plugins***.
2. Na aba ***Available Plugins***, na secção ***Search:*** escrever "*Selenium Module for PHP*".
3. Selecionar o módulo e marcar a *checkbox* ***Install***.
4. Clicar no botão ***Install***.

Finalmente, para correr os testes é preciso configurar o WAMP ativando a extensão *php_curl* no ficheiro "*C:\wamp\bin\php\php5.3.4\php.ini*" através da remoção do caráter ';' que se encontra no início da linha:

```
;extension=php_curl.dll
```

Para além de configurar a *framework* de testes funcionais e unitários sem dependências externas também foi necessário proceder à configuração dos *browsers*, que pode ser consultada no próximo anexo.

Configuração dos *Browsers*

É necessário configurar os *browsers* de forma a que os testes efetuem o *download* automático de ficheiros para os diretórios corretos. A sequência de passos para esse efeito no *Chrome* é a seguinte:

1. Clicar no menu do *Chrome*  na barra de ferramentas do *browser*.
2. Selecionar **Definições**.
3. Clicar em **Mostrar definições avançadas** e deslocar para baixo até à secção **Transferências**.
4. Clicar em **Alterar** e selecionar o diretório onde se pretende que os ficheiros sejam guardados, que no caso da *LabOrders* é: "*NUNODEV**selenium_downloads*".

Relativamente ao *Internet Explorer* os passos são os seguintes:

1. Clicar no botão **Ferramentas**  e, em seguida, clicar em **Ver transferências**.
2. Clicar em **Opções** no canto inferior esquerdo.
3. Clicar em **Procurar** e selecionar o diretório onde se pretende que os ficheiros sejam guardados: "*NUNODEV**selenium_downloads*".

Em relação ao *Firefox* as etapas são as seguintes:

1. Clicar em **Ferramentas** -> **Opções** -> **Geral**.
2. Clicar em **Guardar ficheiros em**.
3. Clicar em **Procurar** e selecionar o diretório onde se pretende que os ficheiros sejam guardados: "*NUNODEV**selenium_downloads*".

O *Firefox* tem a particularidade de confirmar sempre se o utilizador pretende abrir ou guardar o ficheiro, a não ser que o comportamento por defeito seja definido para cada tipo de ficheiro.

Para tal é necessário:

1. Realizar o *download* de um ficheiro com a extensão pretendida (que no caso da empresa são .pdf e .xlsx)
2. Clicar em **Repetir opção para ficheiros deste tipo daqui para a frente**.
3. Clicar em ***OK***.

Em alguns testes surgia o erro "*USERNAME TEXTBOX NOT AVAILABLE*" quando o utilizador já se tinha autenticado manualmente antes de correr os testes. Isto deve-se ao facto do *Firefox* não limpar os *cookies* automaticamente no início e no final de cada teste, ao contrário do que acontece com o *Chrome* e o *Internet Explorer*, e como tal a caixa de texto para o utilizador introduzir o seu *username* não era detetada uma vez que não estava disponível porque a identificação já havia sido realizada previamente através dos *cookies*. Sendo assim é necessário configurar o *Firefox* para este efeito, o que implica a criação do novo perfil "testes" referido anteriormente, através dos seguintes passos:

1. Fechar o *Firefox* caso esteja aberto.
2. Clicar no botão **Iniciar** do *Windows* e selecionar **Executar**.
3. Na caixa de diálogo Executar, escrever:
   ```
   firefox.exe -p
   ```
4. Clicar em ***OK*** para abrir o Gestor de Perfis do *Firefox* (**Figura 25**).

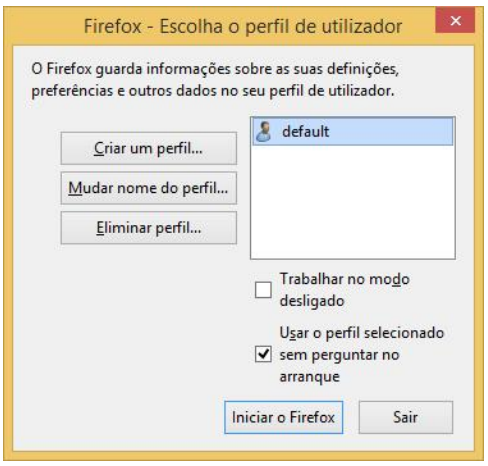

Figura 25 – Gestor de Perfis do *Firefox*

5. Clicar em **Criar um perfil…** no Gestor de Perfis para inicializar o Assistente de criação de Perfil.
6. Clicar em **Seguinte** e introduzir "testes" como o novo nome de perfil.

7. Clicar em **Concluir** (**Figura 26**).

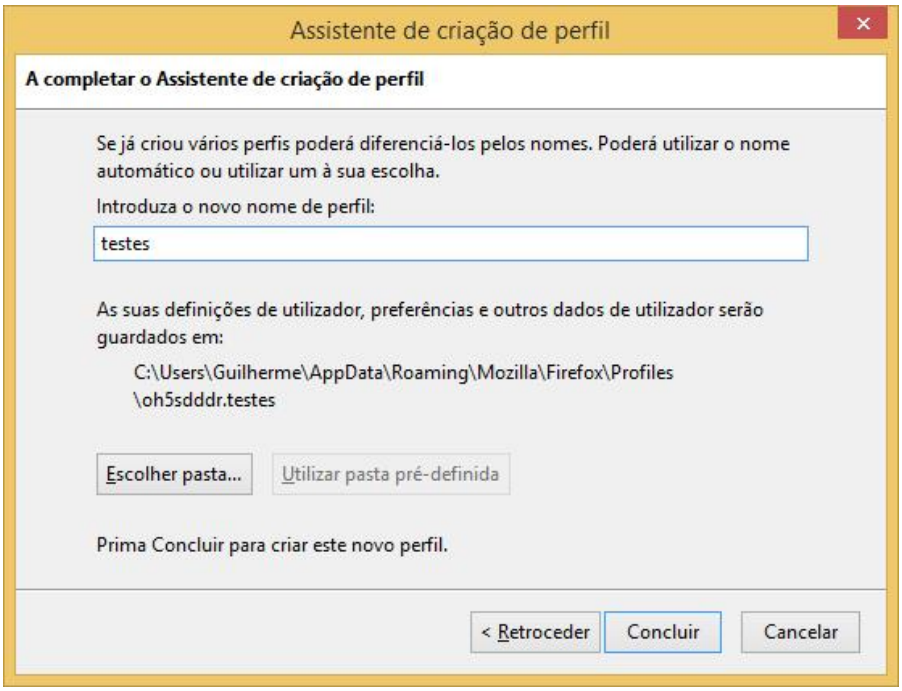

Figura 26 – Criação de um novo perfil do *Firefox*

Com o novo perfil criado resta configurá-lo de forma a que os *cookies* sejam eliminados quando o *Firefox* é fechado. Para isso é preciso:

1. Clicar em **Ferramentas** -> **Opções** -> **Privacidade**.
2. Em "O *Firefox* irá:" selecionar **Usar definições personalizadas para o histórico**.
3. Em "Guardar até:" selecionar **Fechar o *Firefox***.

Clicar em ***OK***.

Configuração do *Kohana*, *SimpleTest* e *Phactory*

A configuração da *framework* de testes unitários foi conseguida através dos seguintes passos:

1. Extrair o *Kohana* para o diretório "*C:\wamp\www\kohana_unit_tests*".
2. Alterar a linha de definição dos erros reportados no ficheiro *index.php*:

```
/**
 * Set the error reporting level. Unless you have a special need, E_ALL is a
 * good level for error reporting.
*/
error_reporting(E_ALL & ~E_DEPRECATED);
```

Para:

```
// Report simple running errors
error_reporting(E_ERROR | E_WARNING | E_PARSE); // Bitwise OR
```

Código 30 – Alteração da definição de erros no *Kohana*

Isto é necessário devido à convenção de nomes do *Kohana*. Os casos de teste têm de ser colocados no diretório "*controllers*\" para poderem ser executados. O *Kohana* requer que as classes dos controladores tenham o mesmo nome do ficheiro do controlador seguido de "*_Controller*", ou seja, para o ficheiro de controlador *vendoruser_test.php*, por exemplo, o nome da classe do controlador tem de ser *Vendoruser_test_Controller*. O *SimpleTest* utiliza uma convenção diferente, a classe de um caso de teste deve começar pela palavra *Test*. Contudo, se a convenção de nomes do *Kohana* não for respeitada (o que nos casos de teste não é possível, tem de se seguir a convenção de nomes do *SimpleTest*) é lançado um aviso. A alteração da linha da definição de erros faz com que esses avisos não sejam exibidos.

Para além disso a versão 2.3.4 do *Kohana* que a empresa utiliza está bastante desatualizada (é uma versão de 2009) e tem algumas funções depreciadas, o que resulta no lançamento de exceções. Mesmo que os testes sejam todos bem-sucedidos, se ocorrer uma exceção devido a uma função depreciada a barra de resultados irá ser mostrada a vermelho e não a verde, como é pretendido nestes casos. A alteração da linha da definição de erros faz com que as exceções de funções depreciadas sejam ignoradas, evitando que interfiram com os resultados dos casos de teste.

3. Exportar a estrutura da base de dados da aplicação:
    a. Clicar no ícone do WAMP e em ***Start All Services*** para iniciar o ***MySQL***.
    b. Aceder ao ***phpMyAdmin***: http://127.0.0.1/phpmyadmin/.
    c. Selecionar a base de dados da aplicação
    d. Clicar no botão **Exportar** -> **Personalizado – exibe todas as opções possíveis** -> **Opções específicas do formato:** -> **estrutura** -> **Executar**.
4. Criar a *mock database*: clicar em **Base de Dados** e em **Criar base de Dados** introduzir o mesmo nome da base de dados da aplicação.

5. Importar a estrutura da base de dados da aplicação: clicar em **Importar**, seguido do botão **Escolher ficheiro** e selecionar o ficheiro .sql gerado no passo 6. c..
6. Efetuar o *download* da versão 1.1.0 do *SimpleTest*: http://sourceforge.net/projects/simpletest/files/simpletest/simpletest_1.1/simpletest_1.1.0.tar.gz/download.
7. Criar o diretório "*C:\wamp\www\kohana_unit_tests\libraries\simpletest*" e extrair para lá o conteúdo do ficheiro *simpletest_1.1.0.tar.gz*.
8. Efetuar o *download* da versão 0.3.1 do *Phactory*: https://github.com/chriskite/phactory/archive/v0.3.1.zip.
9. Criar o diretório "*C:\wamp\www\kohana_unit_tests\libraries\Phactory*" e extrair para lá o conteúdo do ficheiro *phactory-0.3.1.zip*.

De seguida foi feita a integração do *Phactory* com o *Kohana* para ser possível realizar testes com *mock database*. Ambos possuem classes de inflexão e como tal existia uma colisão entre a classe *Inflector* do *Kohana* e a do *Phactory* uma vez que ambas tinham o mesmo nome (*Inflector*). Entende-se por classe de inflexão aquela que permite pluralizar e singularizar nomes [Kohana User Guide]. Isto é necessário porque as convenções do ORM do *Kohana* obrigam a que:

- A tabela onde serão guardadas as informações do modelo tenha o nome do mesmo no plural.
- Os nomes dos modelos e das tabelas tenham de ser escritos em Inglês, por exemplo se o nome do modelo for *person* o nome da tabela deverá ser *people*. Em caso de dúvida pode-se usar a classe *Inflector* para descobrir o singular ou o plural de qualquer palavra:

```
echo Inflector::plural('person');
echo Inflector::singular('people');
```

Código 31 – Exemplo de pluralização e singularização de um nome

Para resolver o problema da colisão alterou-se o nome da classe *Inflector* do *Phactory* para *Inflector_phactory* nos ficheiros
"*C:\wamp\www\kohana_unit_tests\libraries\Phactory\lib\Inflector.php*" e
"*C:\wamp\www\kohana_unit_tests\libraries\Phactory\lib\Phactory\Inflector.php*".

Teste de *Download* de um Documento de Concessão

```
class grantsDocsPage extends basePage {
    // Page Elements
    protected $grantsDocsTableID    = 'itemlist';
    protected $downloadLocation    = 'C:\Users\Nuno\Downloads';
    protected $grantsDocsDownloadFileSpanCSSSelector    = '.ui-icon-
arrowthickstop-1-s';
     protected $grantsDocsGrantDocSpanTagName    = 'span';
     protected $grantsDocsChildRowsCSSSelector    = "tr[id*='file_']";
     protected $grantsDocsTdTagName    = 'td';
     protected $idNumberIndex    = 1;
     protected $grantDocsColumnIndex    = 0;
     protected $actionsColumnIndex    = 4;

    …

    /**
     * Downloads the most recent Grant Doc
     */
    public function downloadMostRecentGrantDoc() {
        // Gets the most recent Grant Doc ID
        $mostRecentGrantDocID = $this->getMostRecentGrantDocID();

        // Gets the most recent Grant Doc child row
        $mostRecentGrantDocChildRow = $this-
>getMostRecentGrantDocChildRow($mostRecentGrantDocID);

        // Calls the getTdWithTextByIndex($mostRecentGrantDocChildRow, $index)
function in order to obtain the Grant / Doc td
        // It’s in the 1st Column of the Grants’ Docs Table (index = 0)
        // Finally, it is stored in the $grantsDocsGrantDocTd variable
        $grantsDocsGrantDocTd = $this-
>getTdWithTextByIndex($mostRecentGrantDocChildRow, $this->
grantDocsColumnIndex);

        // Gets the Grants' Docs Grant / Doc span
        $grantsDocsGrantDocSpan = $grantsDocsGrantDocTd-
>findElementBy(LocatorStrategy::tagName, $this->grantsDocsGrantDocSpanTagName,
"Could not find the Grants' Docs Grant / Doc span");

        // Gets the the Grants' Docs Grant / Doc span Text
        $grantsDocsGrantDocSpanText = $grantsDocsGrantDocSpan->getText();

        // Calls the getTdWithTextByIndex($mostRecentGrantDocChildRow, $index)
function in order to obtain the Actions td
        // It's in the 5th Column of the Grants' Docs Table (index = 4)
        // Finally, it is stored in the $grantsDocsActionsTd variable
        $grantsDocsActionsTd = $this-
>getTdWithTextByIndex($mostRecentGrantDocChildRow, $this->actionsColumnIndex);
        // Gets the Download file span
        $grantsDocsDownloadFileSpan = $grantsDocsActionsTd-
>findElementBy(LocatorStrategy::cssSelector, $this-
>grantsDocsDownloadFileSpanCSSSelector, "Could not find the Grants' Docs
Download file span");
```

```
        // Clicks the Download file span
        // The browser must save the file automatically, this can be achieved
by manually checking the "Do this automatically for files like this from now
on" Checkbox
        $grantsDocsDownloadFileSpan->click();

        if(Configs::BROWSER == 'internet explorer') {
            Runtime.getRuntime().exec("C:\selenium\download.exe");
        }

        // Waits for the file to be downloaded
        sleep(3);

        // Creates the directory string
        // It concatenates the Downloads Location in the Browser Settings with
$grantsDocsGrantDocSpanText (file name)
        $filename = Configs::DOWNLOADS_LOCATION . $grantsDocsGrantDocSpanText;

        // Checks if the file exists in the directory
        // If it doesn't exist, the Test must fail
        if (!file_exists($filename))
            $this->fail('Could not download the Grant Doc');
    }

    …

    /**
     * Returns the most recent Grant Doc ID
     */
    public function getMostRecentGrantDocID() {
        // Creates the $max auxiliary variable and initializes it with 0
        // This variable will be used to store the maximum Grant Doc ID number
        $max = 0;

        // Gets the Grants' Docs Table
        $grantsDocsTable = $this->webdriver->findElementBy(LocatorStrategy::id,
$this->grantsDocsTableID, "Could not find the Grants' Docs Table");
        // Gets the Grants' Docs child Rows
        $grantsDocsChildRows = $grantsDocsTable-
>findElementsBy(LocatorStrategy::cssSelector, $this-
>grantsDocsChildRowsCSSSelector, "Could not find any Grants' Docs child Rows");

        // Iterates over the $grantsDocsChildRows Array
        foreach ($grantsDocsChildRows as $childRow) {
            // Gets the id value
            $grantDocIDAttribute = $childRow->getAttribute('id');

            // Splits the $grantDocIDAttribute by the "_" delimiter and puts
the content in the $grantDocIDAttributeArray Array
            $grantDocIDAttributeArray = explode("_", $grantDocIDAttribute);

            // The Grant Doc ID Number is in the second position (index = 1) of
the $grantDocIDAttributeArray Array
            $grantDocIDNumber = $grantDocIDAttributeArray[$this->idNumberIndex];

            // Casts $grantDocIDNumber to int
            $grantDocIDNumberInt = (int) $grantDocIDNumber;
```

```
            // If $grantDocIDNumberInt is higher than $max it's the maximum
Grant Doc ID number
            // This value is stored in the $max auxiliary variable
            if($grantDocIDNumberInt > $max)
                $max = $grantDocIDNumberInt;
        }

        // Returns the $max auxiliary variable
        return $max;
    }

    /**
     * Returns the most recent Grant Doc child Row
     */
    public function getMostRecentGrantDocChildRow($mostRecentGrantDocID) {
        // Gets the Grants' Docs Table
        $grantsDocsTable = $this->webdriver->findElementBy(LocatorStrategy::id,
$this->grantsDocsTableID, "Could not find the Grants' Docs Table");

        // Creates the Grant Doc most recent child Row ID
        // It concatenates the string 'file_' with the
$mostRecentGrantDocIDText
        $grantsDocsMostRecentChildRowID = 'file_' . $mostRecentGrantDocID;

        // Gets the most recent child Row
        $grantsDocsMostRecentChildRow = $grantsDocsTable-
>findElementBy(LocatorStrategy::id, $grantsDocsMostRecentChildRowID, "Could not
find the Grants' Docs most recent child Row");

        // Returns the most recent child Row
        return $grantsDocsMostRecentChildRow;
    }

    /**
     * Returns the td with the $index index from the
$mostRecentGrantDocChildRow child Row
     */
    public function getTdWithTextByIndex($mostRecentGrantDocChildRow, $index) {
        // Gets the Grants' Docs td's
        $grantsDocsTds = $mostRecentGrantDocChildRow-
>findElementsBy(LocatorStrategy::tagName, $this->grantsDocsTdTagName, "Could
not find any Grants' Docs td");

        // Stores the td with the $index index in the $grantsDocsIndexTd
variable
        $grantsDocsIndexTd = $grantsDocsTds[$index];

        // Returns the $grantsDocsIndexTd variable
        return $grantsDocsIndexTd;
    }
}
```

Código 32 – Classe da página dos documentos das concessões na plataforma

```
class basePage extends PHPUnit_Framework_TestCase {
    /**
     * @var WebDriver
     */

    // Page Elements
    …
    protected $grantsDocsPageLabel = "Grants' Docs";

    …

    /**
     * Navigates to the Grants' Docs Page and returns it
     * @return grantsDocsPage
     */
    public function goToGrantsDocsPage() {
        // Clicks the Grants Docs Navigation Menu Button
        $this->_clickNavMenuButton($this->grantsDocsPageLabel);

        // Returns the Grants Docs Page
        return new grantsDocsPage($this->webdriver);
    }

    …

    /**
     * Clicks the Navigation Menu Button
     */
    private function _clickNavMenuButton($buttonText) {
        // Shows all hidden submenus
        $this->webdriver->execute('$(".navsubmenu").toggle()', array());

        // Gets the Navigation Menu
        $navMenu = $this->webdriver->findElementBy(LocatorStrategy::cssSelector,
'.navmenu', 'Could not find the Navigation menu');

        // Gets the Navigation Menu Button
        $navMenuButton = $navMenu->findElementBy(LocatorStrategy::linkText,
$buttonText, 'Coult not find the Navigation Menu Button "' . $buttonText . '"');

        // Clicks the Navigation Menu Button
        $navMenuButton->click();
    }

    …
}
```

Código 33 – Classe *basePage* do teste de *download* de um documento de concessão

```
/**
 * Test … Grant Doc downloading …
 */

class grantDocsTest extends baseTest {
    …

    /**
     * Tests if the Grant Doc is correctly downloaded
     */
    public function testGrantDocDownloading() {
        // Logs in and navigates to the Grants' Docs Page
        $grantsDocsPage = $this->loginAndGoToGrantsDocs();

        // Downloads the most recent Grant File
        $grantsDocsPage->downloadMostRecentGrantDoc();
    }

    /**
     * Logs in, navigates to the Grant's Docs Page and returns it
     * @return grantsDocsPage
     */
    public function loginAndGoToGrantsDocs() {
        // Creates a new Login Page and stores it in the $loginPage variable
        $loginPage = new logInPage($this->webdriver);

        // Logs in
        $basePage = $loginPage->goodLoginUser(Configs::IBET_LABPI_USERNAME,
Configs::USERSPW);

        // Navigates to the Grants' Docs Page
        $grantsDocsPage = $basePage->goToGrantsDocsPage();

        // Returns the Grants' Docs Page
        return $grantsDocsPage;
    }
}
```

Código 34 – Classe de teste do *download* de um documento de concessão

Teste do Conteúdo de Uma Ordem de Encomenda

```
class labOrdersPage extends basePage {
    // Page Elements
    …
    protected $orderHistoryTableID    = 'itemlist';
    protected $trClassName    = 'parent';
    protected $vendorNameDivClassName    = 'parent_title';
    protected $vendorNameSpanTagName    = 'span';
    protected $priceSpanClassName    = 'price_column_element';
    protected $checkboxClassName    = 'js_row_checkbox';
    protected $pdfLinkClassName    = 'generate_pdf_report';
    protected $childTrCSSSelector    = '.child.tr_size_50.trstrongodd';
    protected $productNameDivClassName    = 'product_description_container';

    …

    /**
     * Returns the tr from the first Order
     */
    public function getTr() {
        // Gets the Summary of requisition Table
        $orderHistoryTable = $this->webdriver-
>findElementBy(LocatorStrategy::id, $this->orderHistoryTableID, 'Could not find
the Order history Table');

        // Gets the tr from the first Order and stores it in the $tr variable
        $tr = $orderHistoryTable->findElementBy(LocatorStrategy::className,
$this->trClassName, "Could not find the tr from the first Order");

        // Returns $tr
         return $tr;
    }

    /**
     * Returns the Vendor name from the first Order
     */
    public function getVendorName() {
        // Gets the Vendor name tr
        $VendorNameTr = $this->getTr();

        // Gets the Vendor name div
        $vendorNameDiv = $VendorNameTr-
>findElementBy(LocatorStrategy::className, $this->vendorNameDivClassName,
"Could not find the Vendor name div");

        // Gets the Vendor name span
        $vendorNameSpan = $vendorNameDiv-
>findElementBy(LocatorStrategy::tagName, $this->vendorNameSpanTagName, "Could
not find the Vendor name span");

        // Gets the Vendor name span text
        $vendorName = $vendorNameSpan->getText();

        // Converts a string with ISO-8859-1 characters encoded with UTF-8 to
single-byte ISO-8859-1
        $vendorName = utf8_decode($vendorName);
```

```
        // Returns $vendorName
        return $vendorName;
    }

    /**
     * Returns the Price from the first Order
     */
    public function getPrice() {
        // Gets the Price tr
        $priceTr = $this->getTr();

        // Gets the Price span's
        $priceSpan = $priceTr->findElementBy(LocatorStrategy::className, $this-
>priceSpanClassName, "Could not find the Price span");
        // Gets the Price span Text and stores it in the $price variable
        $price = $priceSpan->getText();

        // Removes the '€' symbol from $price
        $price = str_replace('€', '', $price);

        // Returns $price
        return $price;
    }

    /**
     * Returns the Checkbox from the first Order
     */
    public function getCheckbox() {
        // Gets the Order Checkbox tr
        $orderCheckboxTr = $this->getTr();

        // Gets the Order Checkbox and stores it in the $orderCheckbox variable
        $orderCheckbox = $orderCheckboxTr-
>findElementBy(LocatorStrategy::className, $this->checkboxClassName, "Could not
find the Order Checkbox");

        // Returns $orderCheckbox
        return $orderCheckbox;
    }

    /**
     * Returns the Product name from the first Order
     */
    public function getProductName() {
        // Gets the Summary of requisition Table
        $orderHistoryTable = $this->webdriver-
>findElementBy(LocatorStrategy::id, $this->orderHistoryTableID, 'Could not find
the Order history Table');

        // Gets the child tr from the first Order and stores it in the $childTr
variable
        $childTr = $orderHistoryTable-
>findElementBy(LocatorStrategy::cssSelector, $this->childTrCSSSelector, "Could
not find the child tr from the first Order");

        // Gets the Product name div
        $productNameDiv = $childTr->findElementBy(LocatorStrategy::className,
$this->productNameDivClassName, 'Could not find the Product name div');

        // Gets the Product name div Text and stores it in the $productName
variable
        $productName = $productNameDiv->getText();
```

```
        // Returns $productName
        return $productName;
    }

    /**
     * Downloads the .pdf file from the first Order
     */
    public function downloadPDF() {
        // Gets the PDF Link from the first Order
        $pdfLink = $this->webdriver->findElementBy(LocatorStrategy::className,
$this->pdfLinkClassName, "Could not find the PDF Link from the first Order");

        // Clicks the PDF Link
        $pdfLink->click();
        // Waits for the file to be downloaded
        sleep(3);
    }
}
```

Código 35 – Classe da página das encomendas

```
class basePage extends PHPUnit_Framework_TestCase {
    /**
     * @var WebDriver
     */
    Protected $webdriver;

    // Page Elements
    …
    protected $labOrdersButtonLink = 'Lab Orders';

    …

    /**
     * Navigates to the LabOrders Page and returns it
     * @return labOrdersPage
     */

    public function goToLabOrders() {
        // Gets the LabOrders Link
        $labOrdersLink = $this->webdriver-
>findElementBy(LocatorStrategy::linkText, $this->labOrdersButtonLink, 'Could
not find the LabOrders Link');

        // Clicks the LabOrders Link
        $labOrdersLink->click();

        // Returns the LabOrdersPage
        return new labOrdersPage($this->webdriver);
    }

    …
}
```

Código 36 – Classe *basePage* do teste do conteúdo de uma ordem de encomenda

```
require_once '../lib/pdf2text/pdf2text.php';

class receivalsPDFTest extends baseTest {
    …
    /**
     * Tests if the generated .pdf file of the first order in the Order history
Table has the correct Vendor name, Product name and Price
     */
    public function testCreateOrderPDF() {
        if(Configs::BROWSER == 'internet explorer') {
            // Stop here and mark this test as incomplete
            $this->markTestIncomplete('Test download Internet Explorer');
        }
        else {
            // Logs in
            $basePage = $this->goodLogin(Configs::IBET_LABPI_USERNAME);

            // Navigates to the Lab Orders Page
            $labOrdersPage = $basePage->goToLabOrders();

            // Gets the Vendor name from the first Order in the Order history
Table
            $vendorName = $labOrdersPage->getVendorName();

            // Gets the Product name from the first Order in the Order history
Table
            $productName = $labOrdersPage->getProductName();

            // Splits the Product name and the Product reference
            $productNameArray = explode("#", $productName);

            // The Product name is in the first position (index = 0) of the
$productNameArray
            $productNameSplit = $productNameArray[0];

            // Converts a string with ISO-8859-1 characters encoded with UTF-8
to single-byte ISO-8859-1
            $productNameSplit = utf8_decode($productNameSplit);

            // Gets the Price from the first Order in the Order history Table
            $price = $labOrdersPage->getPrice();

            // Gets the Order Checkbox
            $orderCheckbox = $labOrdersPage->getCheckbox();

            // Clicks the Order Checkbox
            $orderCheckbox->click();

            // Downloads the .pdf file from the first Order in the Order
history Table
            $labOrdersPage->downloadPDF();

            // Gets the most recent filename in the download directory
            $filenameDir = $this->getMostRecentFilename();
```

```
            // It concatenates the Downloads Location in the Browser Settings
with $filenameDir
            $directory = Configs::DOWNLOADS_LOCATION . $filenameDir;

            // Gets the .pdf file content
            $pdfContent = pdf2text($directory);

            // Verifies if the Vendor name exists in the .pdf file
            $this->assertContains($vendorName, $pdfContent, "The generated .pdf
file doesn't contain the Vendor name: " . $vendorName);

            // Verifies if the Product name exists in the .pdf file
            $this->assertContains($productNameSplit, $pdfContent, "The
generated .pdf file doesn't contain the Product name: " . $productNameSplit);

            // Verifies if the Price exists in the .pdf file
            $this->assertContains($price, $pdfContent, "The generated .pdf file
doesn't contain the Price: " . $price);
        }
    }

    /**
     * Returns the most recent filename in the download directory
     */
    private function getMostRecentFilename() {
        $path = Configs::DOWNLOADS_LOCATION;

        $latest_ctime = 0;
        $latest_filename = '';

        $d = dir($path);
        while(false !== ($entry = $d->read())) {
            $filepath = "{$path}/{$entry}";
            // Could do also other checks than just checking whether the entry
is a file
            if(is_file($filepath) && filectime($filepath) > $latest_ctime) {
                $latest_ctime = filectime($filepath);
                $latest_filename = $entry;
            }
        }

        // now $latest_filename contains the filename of the file that changed
last

        // Returns $latest_filename
        return $latest_filename;
    }
}
```

Código 37 – Classe de teste do conteúdo de uma ordem de encomenda

Caso de Teste dos Serviços

```
class catalogPage extends basePage {
    …

    /**
     * Searches for a specific product and returns it
     * @param string $searchQuery
     * @return WebElement
     */
    public function searchProduct($searchQuery) {
        // Searches by $searchQuery
        $this->search($searchQuery);

        // Gets all the child Products and stores them in the $A_childProducts
Array
        $A_childProducts = $this->getChildProducts();

        // Gets the Product that matches the $searchQuery
        $sku_element = $this->findAndReturnElementByText($A_childProducts,
$searchQuery, Could not find the Product');

        // Returns the Product
        return $sku_element;
    }

    …
}
```

Código 38 – Classe da página do catálogo

```
class basePage extends PHPUnit_Framework_TestCase {
    /**
     * @var WebDriver
     */
    protected $webdriver;

    // Page Elements
    …
    protected $catalogButtonLink    = 'Catalog';
    protected $listERPBudgetLink    = 'Budgets';

    …

    /**
     * Navigates to the Catalog Page and returns it
     * @return catalogPage
     */
    public function goToCatalog() {
        // Gets the Catalog Link
        $catalogLink = $this->webdriver-
>findElementBy(LocatorStrategy::linkText,   $this->catalogButtonLink, 'Could
not find the Catalog Link');

        // Clicks the Catalog Link
        $catalogLink->click();

        // Returns the Catalog Page
        return new catalogPage($this->webdriver);
    }

    …

    /**
     * Navigates to the List ERP Budget Page and returns it
     * @return listERPBudgetPage
     */
    public function goToListERPBudgetPage() {
        // Shows all hidden submenus
        $this->webdriver->execute('$(".navsubmenu").toggle()', array());

        // Gets the ERP Budget Link
        $listERPBudgetLink = $this->webdriver-
>findElementBy(LocatorStrategy::linkText,   $this->listERPBudgetLink, 'Could
not find the List ERP Budget Link');

        // Clicks the ERP Budget Link
        $listERPBudgetLink->click();

        // Returns the ERP Budget Page
        return new listERPBudgetPage($this->webdriver);
    }

    …
}
```

Código 39 – Classe *basePage* do caso de teste dos serviços

```
class aaSphinxAndServicesTest extends baseTestIGC {
    // Page Elements
    protected $textProductReference   = 'gloves';

    /**
     * Tests the product search (Sphinx)
     */
    public function testSearch() {
        // Logs in
        $basePage = $this->goodLogin(Configs::IBET_LABPI_USERNAME);

        // Navigates to the Catalog Page
        $catalogPage = $basePage->goToCatalog();

        // Searches for the Product with the $textProductReference
        $catalogPage->customSearch($this->textProductReference);
    }

    /**
     * Signs up filling the required fields (SMTP)
     */
    public function testSignUpWithFieldsRequired() {
        // Gets the Sign Up Page Form
        $signUpPageForm = $this->getSignUpPageForm();

        // Fills the Sign Up Page Form with the required fields
        $signUpPageForm->fillFormWithRequiredFields('Selenium User',
'teste1@laborders.com', 'Institution Test');

        // Submits the Sign Up Page Form
        $signUpPageForm->submit();

        // The success message must be present
        $signUpPageForm->checkIfSuccessMessageIsPresent();
    }

    /**
     * Tests if the ERP Budget appears (SoapUI)
     */
    public function testERPBudgetAppears() {
        // Logs in as IGC Lab PI
        $catalogPage = $this->LoginIGCLabPi();

        // Goes to the ListERPBudgetPage
        $listERPBudgetPage = $catalogPage->goToListERPBudgetPage();

        // Expands the Grants
        $listERPBudgetPage->expandColapseGrants();

        // Validates if the budget graphics appear
        $listERPBudgetPage->validateGraphicsAppear()
    }

    /**
     * Returns the Sign Up Page Form
     */
    private function getSignUpPageForm() {
        // Gets the Login Page
        $loginpage= new logInPage($this->webdriver);
```

```
        // Gets the Sign Up Page Form
        $signUpPageForm = $loginpage->goToSignUpPage();

        // Returns the Sign Up Page Form
        return $signUpPageForm;
    }
}
```

Código 40 – Classe do caso de teste dos serviços

Aplicação de Testes (Prova de Conceito)

```
class Vendor_Model extends ORM {
    protected $has_many = array('users');

    public function get_users() {
        return $this->users;
    }

    public function get_vendor_names() {
        $A_vendor_names = array();

        $A_vendors = ORM::factory('vendor')->find_all();

        foreach($A_vendors as $vendor) {
            $vendor_name = $vendor->name;

            $A_vendor_names[] = $vendor_name;
        }

        return $A_vendor_names;
    }

    public static function get_by_id($id) {
        $vendor = ORM::factory('vendor', $id);

        return $vendor;
    }
}
```

Código 41 – Modelo do fornecedor (*vendor*)

```
class User_Model extends ORM {
    protected $belongs_to = array('vendor');

    public function get_name() {
        $name = $this->name;

        return $name;
    }

    public function get_last_name() {
        $name = $this->get_name();

        $A_name = explode(' ', $name);

        $last_name = array_pop($A_name);

        return $last_name;
    }

    public function get_vendor() {
        $vendor = $this->vendor;

        return $vendor;
    }

    public function get_number_users() {
        $A_users = ORM::factory('user')->find_all();

        $n_users = count($A_users);

        return $n_users;
    }
}
```

Código 42 – Modelo do utilizador (*user*)

```
<?php defined('SYSPATH') OR die('No direct access allowed');

class Vendoruser_Controller extends Template_Controller {
    public $template = 'vendoruser_content';

    public function list_all() {
        $vendor = ORM::factory('vendor');

        $A_vendor_names = $vendor->get_vendor_names();

        $this->template->A_vendor_names = $A_vendor_names;

        $A_vendors = ORM::factory('vendor')->find_all();

        $this->template->A_vendors = $A_vendors;
    }
}
?>
```

Código 43 – Controlador da aplicação de testes

```
<html>
    <head>
        <title>LabOrders</title>
    </head>
    <body>
        <?php
        if(!empty($_POST)) {
        ?>
            <h2 align="center"><?php echo $_POST['vendorName']; ?> users</h2>
            <table align="center" border="1" cellpadding="10" cellspacing="0">
                <tr>
                    <th>id</th>
                    <th>vendor_id</th>
                    <th>name</th>
                </tr>

                <?php
                foreach($A_vendors as $vendor) {
                    if($vendor->name == $_POST['vendorName']) {
                        $A_users = $vendor->get_users();
                        foreach($A_users as $user) {
        ?>
                            <tr>
                                <td align="center"><?php echo $user-
>id; ?></td>
                                <td align="center"><?php echo $user-
>vendor_id; ?></td>
                                <td align="center"><?php echo $user-
>name; ?></td>
                            </tr>
        <?php
                        }
                    }
                }
                ?>
            </table>
        <?php
        }
        else
        {
        ?>
            Vendors
            <form method="post">
            <select name="vendorName">
            <?php
            foreach($A_vendor_names as $vendor_name) {
            ?>
                <option value="<?php echo $vendor_name; ?>"><?php echo
$vendor_name; ?></option>
            <?php
            }
            ?>
                <input type="submit" value="OK">
            </select>
            </form>
        <?php
        }
        ?>
    </body>
<html>
```

Código 44 – Vista da aplicação de testes

Caso de Teste da Obtenção de Utilizadores

```
require_once('\libraries\simpletest\autorun.php);
require_once('\libraries\Phactory\lib\Phactory.php);
require_once('\models\unit_tests\models\vendor.php);
require_once('\models\unit_tests\models\user.php');

// this call generates a class called 'Vendor_Model'
// with the same interface as 'Vendor_Model'
Mock::generate('Vendor_Model');

class Vendoruser_test extends UnitTestCase {

    …

    public function test_get_users_mock_database() {
        $this->create_vendors_mock_database();

        $A_users_phactory_rows = $this->create_users_mock_database();

        $A_vendors_mock_database = $this->get_vendors_mock_database();

        $A_users1 = $A_vendors_mock_database[0]->get_users();

        $A_users2 = $A_vendors_mock_database[1]->get_users();

        for($i = 0; $i <= 1; $i++) {
            $this->assertEqual($A_users_phactory_rows[$i]->vendor_id,
$A_users1[$i]->vendor_id);
            $this->assertEqual($A_users_phactory_rows[$i]->name, $A_users1[$i]-
>name);
        }

        for($i = 2; $i <= 6; $i++) {
            $this->assertEqual($A_users_phactory_rows[$i]->vendor_id,
$A_users2[$i]->vendor_id);
            $this->assertEqual($A_users_phactory_rows[$i]->name, $A_users2[$i]-
>name);
        }
    }

        public function create_vendors_mock_database() {
            // create a row in the db with id = 1 and name = 'Alfagene'
            $A_vendors_phactory_rows[] = Phactory::create('vendor', array('id'
=> 1, 'name' => 'Alfagene'));
            // create a row in the db with id = 2 and name = 'Via Athena'
            $A_vendors_phactory_rows[] = Phactory::create('vendor', array('id'
=> 2, 'name' => 'Via Athena'));

            return $A_vendors_phactory_rows;
        }

        public function get_vendors_mock_database() {
            for($i = 1; i <= 2; $i++) {
                $A_vendors_mock_database[] = ORM::factory('vendor', $i);
            }

            return $A_vendors_mock_database;
        }
```

```
        public function create_users_mock_database() {
            // create a row in the db with id = 1, vendor_id = 1 and name =
'Luis Guilherme', and store a Phactory_Row object
            $A_users_phactory_rows[] = Phactory::create('user', array('id' => 1,
'vendor_id' => 1, 'name' => 'Luis Guilherme'));
            // create a row in the db with id = 2, vendor_id = 1 and name =
'Gomes Albuquerque', and store a Phactory_Row object
            $A_users_phactory_rows[] = Phactory::create('user', array('id' => 2,
'vendor_id' => 1, 'name' => 'Gomes Albuquerque'));
            // create a row in the db with id = 3, vendor_id = 2 and name =
'Santos Castelo', and store a Phactory_Row object
            $A_users_phactory_rows[] = Phactory::create('user', array('id' => 3,
'vendor_id' => 2, 'name' => 'Santos Castelo'));
            // create a row in the db with id = 4, vendor_id = 2 and name =
'Ana Maria', and store a Phactory_Row object
            $A_users_phactory_rows[] = Phactory::create('user', array('id' => 4,
'vendor_id' => 2, 'name' => 'Ana Maria'));
            // create a row in the db with id = 5, vendor_id = 2 and name =
'José António', and store a Phactory_Row object
            $A_users_phactory_rows[] = Phactory::create('user', array('id' => 5,
'vendor_id' => 2, 'name' => 'José António'));
            // create a row in the db with id = 6, vendor_id = 2 and name =
'Susana Cristina', and store a Phactory_Row object
            $A_users_phactory_rows[] = Phactory::create('user', array('id' => 6,
'vendor_id' => 2, 'name' => 'Susana Cristina'));
            // create a row in the db with id = 7, vendor_id = 2 and name =
'Luis Filipe', and store a Phactory_Row object
            $A_users_phactory_rows[] = Phactory::create('user', array('id' => 7,
'vendor_id' => 2, 'name' => 'Luis Filipe'));

            return $A_users_phactory_rows;
    }
}
```

Código 45 – Caso de teste da obtenção de utilizador com *mock objects* e *mock database*

Sites em *WordPress*

Foram desenvolvidos três *sites* em *WordPress* como trabalho extra, conforme foi referido previamente:

- ENEB – http://eneb.pt/2015/
- GraPE – http://grape.pt/2014/
- *Drosophila Meeting* – http://drosophilameeting.pt/2015/

A ligação ao servidor *File Transfer Protocol* (FTP, protocolo de transferência de ficheiros através da *Internet*) e o envio dos ficheiros foram feitos com o *FileZilla* mas pode ser usado o *cPanel* ou qualquer cliente FTP. Também foi criado o diretório *"\public_html\2015\"* de forma a suportar múltiplos *sites* no mesmo domínio (o *site* do próximo ano será colocado no diretório *"\public_html\2016\"*). O redirecionamento das páginas foi conseguido através de um ficheiro index.php no diretório "*public_html*" com o seguinte conteúdo:

```
header("Location: http://eneb.pt/2015/");
```

Neste caso, quando o utilizador acede a http://eneb.pt/ é redirecionado por http://eneb.pt/2015/. O procedimento para os outros *sites* é o mesmo, só mudam os endereços.

A *LabOrders* pretendia *sites* com um menu de hiperligações no topo e múltiplas secções, sendo que o clique numa hiperligação deve fazer o *site* deslizar automaticamente até à secção respetiva. Como tal foi utilizado o tema *SKT Parallax Me*: http://www.sktthemes.net/themes/skt_parallax_me/, um *layout* de uma única página (apesar de ser possível criar outras páginas sem o efeito de paralaxe [movimentação automática entre secções despoletada por cliques nas hiperligações]) e gratuito (até 10 secções).

Foram efetuadas várias alterações ao tema. Clicando em ***Appearance*** -> ***Theme Options*** é possível fazer o *upload* do logotipo e do ícone de favoritos, que foi gerado no *site* http://favicon-generator.org.

O separador ***Basic Settings*** é onde se introduz a informação que se quer que apareça no *site*. Pode-se selecionar o número de secções (até ao máximo de 10). O ***Menu Title*** é o texto que vai ser exibido no menu de hiperligações do topo. Pode ser deixado em branco caso não se pretenda que esta secção seja uma entrada do menu.

O *slider* (elemento *web* que faz um conjunto de imagens deslizar horizontalmente ou verticalmente) da página principal do tema teve de ser removido porque cortava as figuras no topo durante o efeito de paralaxe. Foi substituído pelo *Meta Slider* como irá ser descrito posteriormente quando forem abordados os *plugins* utilizados. Para remover o *slider* da página principal do tema foram eliminadas as seguintes linhas de código do ficheiro *header.php*:

```
<?php if((of_get_option('innerpageslider', true) != 'hide') || is_home() ||
is_front_page()) { ?>
<section>
    <div class ="feature">
        <div class="slider_text">
            <div id="slidecaption"></div>
        </div>
        <div class="control-nav">
            <a id="prevslide" class="load-item"><i class="font-icon-arrow-
simple-left"></i></a>
            <a id="nextslide" class="load-item"><i class="font-icon-arrow-
simple-right"></i></a>
            <ul id="slide-list"></ul>
            <a id="nextsection" href="#work"><i class="font-icon-arrow-simple-
down"></i></a>
        </div>
    </div><!-- -/feature -->
</section>
<?php } ?>
```

Código 46 – *Slider* original da página principal do tema

Para incluir o texto "*Powered by LabOrders*" no lado esquerdo do rodapé do *site*, o seguinte código foi substituído no ficheiro *footer.php*:

```
echo esc_url(of_get_option('footertext', true));
```

Por:

```
echo "Powered by";
?>
<a href=https://www.laborders.com style="color: #FFFFFF"
target="_blank">LabOrders</a>
<?php
```

Código 47 – Inclusão do texto "*Powered by LabOrders*" no rodapé do *site*

No separador ***Social Settings*** é possível definir os perfis das redes sociais (*Facebook*, *Google Plus*, *Youtube*, *Twitter*, *LinkedIn*) que serão mostrados no rodapé do *site*. Caso não se queira incluir algum destes perfis, basta não incluir a hiperligação e o ícone respetivo não será exibido.

Clicando em ***Appearance*** -> ***Customise*** é possível definir o título e a mensagem de boas vindas do *site*.

Para remover a caixa "*Leave a Reply*" (uma vez que se trata de um *site* e não de um fórum e como tal não se pretende que os utilizadores deixem comentários diretos ao conteúdo) é necessário clicar em ***Settings*** -> ***Discussion***, desmarcar a *checkbox* "*Allow people to post comments on new articles*" e por fim clicar no botão ***Save Changes***. Isto deve ser feito antes de se criar qualquer página no *site*. As definições podem ser alteradas para páginas individuais.

O separador das secções localizado em "*\public_html\2015\wp-content\themes\skt-parallaxme\images\heading-splitter.png*" foi editado utilizando o GIMP (mas pode ser usado outro *software* de edição de imagem como o *Photoshop*) de forma a exibir os logotipos respetivos ao evento de cada *site*.

O logotipo do *site* (que aparece no menu de hiperligações) não deve ter mais do que 80 pixeis de altura, caso contrário irá ocultar os títulos das secções.

No caso do *site* do Grape foi ainda requisitada a remoção da hiperligação *Home* no menu e a deslocação da hiperligação de registo para o lado direito. Para isso foi preciso substituir o seguinte código no ficheiro *header.php*:

```
echo skt_parallaxme_str_lreplace('</ul>', $menu_list . '</ul>',
str_replace(array('<div class="menu">', '</div>'), '', $navMenu));
```

Por:

```
echo skt_parallaxme_str_lreplace('<ul>', '<ul>' . $menu_list,
str_replace(array('<div class="menu">', '</div>'), '', $navMenu));
```

Código 48 – Remoção da hiperligação *Home* do menu e deslocação da hiperligação de registo para o lado direito

Para incluir o *poster* do evento foi efetuado o *upload* da imagem respetiva e o seguinte código foi adicionado no final do ficheiro *header.php*:

```
<div align="center">
    <img src=http://grape.pt/2014/wp-content/uploads/cartaz-grape2014-Final.jpg
alt="" />
</div>
```

Código 49 – Inclusão do *poster* do evento

A última modificação no tema é a alteração do formato das hiperligações das páginas que foi conseguida clicando em ***Pages*** -> ***All Pages*** e selecionando a página pretendida. É necessário então selecionar a estrutura pretendida em ***Permalink Settings***. Alterando a opção por defeito (*Page ID*, número único identificador da página) para a opção *Post Name* (nome da publicação), a hiperligação irá conter o título da página em vez do *ID* (por exemplo http://grape.pt/2014/registo/ em vez de http://grape.pt/2014/?p=13).

De seguida irão ser listados os *plugins* utilizados e as respetivas funcionalidades.

O ***WP Google Fonts*** foi utilizado para adicionar tipos de letra do diretório da *Google* ao *site*:

1. Clicar em ***Settings*** -> ***Google Fonts***.
2. Selecionar os tipos de letra.

3. Clicar em ***Show Options***.
4. Escolher os tipos de letra pretendidos (tamanho, negrito, itálico…) e os elementos aos quais atribuir o tipo de letra (todas as etiquetas *body* (corpo do conteúdo), títulos [etiquetas h1 a h6], parágrafos [etiquetas p], …). O tipo de letra nos editores de página/secção irá ser substituído pelo do *plugin* ***WP Google Fonts*** sempre que se utilizarem os elementos especificados.
5. Escolher o conjunto de carateres desejado (Latino, Cirílico, …).
6. Clicar em ***save all fonts***.

O ***Meta Slider*** foi usado para exibir um conjunto personalizado de imagens no *site*:

1. Aceder ao ***Meta Slider*** no painel de controlo do *WordPress*.
2. Clicar em ***Add Slide***.
3. Selecionar uma imagem da Biblioteca de Media ou efetuar o *upload* de uma nova (***Upload Files***).
4. No separador ***Settings*** é possível escolher o estilo (aspeto visual) do *slider* (*Nivo Slider*, *Flex Slider*, *R. Slider*, *Coin Slider*) e configurar alguns parâmetros (Width e Height [largura e altura, uma vez que se trata de um *slider* responsivo é recomendado definir as mesmas dimensões das imagens], efeito [*Fade* – Desvanecer, *Slide In Right* – Deslizar para a direita]), tema [*Dark* – Escuro, *Light* – Claro, *Bar* – Barra personalizada], …).
5. Clicar em ***Save***.
6. Pode-se então incluir o *slider* nas páginas/secções do *site* colando o código respetivo, por exemplo:

```
[metaslider id=204]
```

Ou incluindo o código no tema (editando os ficheiros .css):

```
echo do_shortcode(“[metaslider id=204]”);
```

O ***Simple Custom CSS*** adiciona estilos CSS personalizados que substituem os estilos padrão do tema e dos restantes *plugins*:

1. Clicar em ***Appearance*** -> ***Custom CSS***.
2. Escrever os estilos CSS.
3. Clicar em ***Update Custom CSS***. Os estilos irão ser aplicados ao *site*.

O ***Dynamic To Top*** adiciona um botão automático e dinâmico para fazer o *site* deslizar facilmente de volta ao topo:

1. Clicar em ***Appearance*** -> ***To Top***.
2. Configurar o comportamento e o aspeto visual do botão.
3. Clicar em ***Save Changes***. O botão irá aparecer no *site*.

O tema estava a exibir o título do *site* 2 vezes. Para resolver este problema foi instalado o ***Yoast SEO Plugin***:

1. Clicar em ***SEO*** -> ***Titles & Metas***.
2. Clicar no separador ***Home***.
3. Substituir o texto na caixa de texto ***Title template:*** com o título desejado para o *site*.
4. Clicar em ***Save Changes***. O título do *site* irá ser corretamente exibido.

O ***Contact Form 7*** adiciona um formulário de registo que envia uma mensagem de correio eletrónico sempre que um utilizador se regista:

1. Clicar em ***Contact*** -> ***Add New***.
2. Selecionar o idioma, por exemplo "*Portuguese (Portugal)*" e clicar em ***Add New***.
3. Gerar as etiquetas pretendidas para o formulário (para etiquetas de *Completely Automated Public Turing test to tell Computers and Humans Apart* [CAPTCHA, um teste *online* baseado num desafio-resposta para distinguir humanos de computadores ou programas automatizados], o *plugin* ***Really Simple CAPTCHA*** deve ser previamente instalado).
4. Copiar e colar os códigos respetivos às etiquetas na janela de formulário do lado esquerdo, por exemplo *[text* your-name]*.
5. Configurar o formato da mensagem de correio eletrónico (campos *To:* [Para:] *From:* (De:) e *Subject* [Assunto], bem como o *Message body* [corpo ou conteúdo da mensagem]).
6. Clicar em ***Save***.
7. Copiar e colar o código do formulário na página/secção onde se pretende incluir o mesmo, por exemplo:

```
[contact-form-7 id="23" title="Registo"]
```

O ***Flamingo*** armazena na base de dados a informação submetida através do formulário:

- Clicar em ***Flamingo*** -> ***Address Book*** de forma a aceder ao nome e endereço de correio eletrónico de todos os utilizadores que submeteram o formulário.
- Clicar em ***Flamingo*** -> ***Inbound Messages*** para aceder a todos as mensagens que foram submetidas através do formulário.

O ***Exec-PHP*** executa código PHP em páginas/secções:

- Basta instalar e ativar o *plugin* e ficar-se-á preparado para executar código PHP em páginas/secções.
- O código seguinte foi escrito na página onde se pretendia exibir a informação dos utilizadores registados (neste exemplo apenas foi incluído o nome já que o raciocínio é o mesmo para os outros campos da tabela [instituição/empresa, cidade e país], só se alteram as consultas para obtenção dos dados):

```
global $wpdb;
/* wpdb class should not be called directly. global $wpdb variable is an
instantiation of the class already set up to talk to the WordPress database */

$result_id = $wpdb->get_results("SELECT ID From gr_posts WHERE post_type =
'flamingo_inbound'");
/* multiple row results can be pulled from the database with the get_results
function which outputs an object that is stored in $result */

echo '
<table id="hor-minimalist-b">
    <tr>
        <th>
        NOME
        </th>
        …
    </tr>
';

foreach($result_id as $row_id) {
    $id = $row_id->ID;

    echo '
    <tr>';

    $result_nome = $wpdb->get_results("SELECT meta_value FROM gr_postmeta WHERE
post_id = '$id' AND meta_key = '_field_nome'")

    foreach($result_nome as $row_nome) {
        $nome = $row_nome->meta_value;

        echo '
            <td>';
            echo $nome;
            echo '
            </td>';
    }

    …

    echo '
    </tr>';
}
/* Print the contents of $result looping through each row returned in the
result */

echo '
</table>
';
```

Código 50 – Tabela com a informação dos utilizadores registados

Todos os *plugins* foram instalados diretamente através do painel de controlo do *WordPress* (***Plugins*** -> ***Add New*** -> ***Search Plugins*** -> ***Install Now***).

As animações de *scroll* (à medida que se desliza a página) foram obtidas utilizando o *plugin* ***WOW.js*** (http://mynameismatthieu.com/WOW/) e a biblioteca de animações ***animate.css***:

- Efetuar o *download* do ***animate.css***: https://raw.githubusercontent.com/daneden/animate.css/master/animate.min.css.
- Criar um diretório "*css*" no diretório do tema (se ainda não existir) e realizar o *upload* do ficheiro *animate.min.css* utilizando o FileZilla (também se pode usar o *cPanel* ou outros clientes de FTP).
- Efetuar o download do ***WOW.js***: https://raw.githubusercontent.com/matthieua/WOW/master/dist/wow.min.js.
- Criar um diretório "*js*" no diretório do tema (se ainda não existir) e realizar o upload do ficheiro *wow.min.js*.
- Adicionar o código seguinte no ficheiro *functions.php* do tema:

```
// Do NOT include the opening PHP tag

// Enqueue animate.css and WOW.js
add_action('wp_enqueue_scripts', 'sk_enqueue_scripts');
function sk_enqueue_scripts() {
    wp_enqueue_style('animate', get_stylesheet_directory_uri() .
'\css\animate.min.css');

    wp_enqueue_style('wow', get_stylesheet_directory_uri() . '\js\wow.min.js',
array(), '', true);
}

// Enqueue script to activate WOW.js
add_action('wp_enqueue_scripts', 'sk_wow_init_in_footer');
function sk_wow_init_in_footer() {
    add_action('print_footer_scripts', 'wow_init');
}

// Add Javascript before </body>
function wow_init() { ?>
    <script type="text/javascript">
        new WOW().init();
    </script>
<?php }
```

Código 51 – Configuração do plugin *WOW.js* e da biblioteca *animate.css*

O *WOW.js* está agora preparado para animar qualquer elemento que tenha uma classe *wow* e uma classe de animação específica do *animate.css*, por exemplo:

```
<div class="wow bounceInUp">Content to Reveal Here</div>
```

As animações que se seguem foram usadas no *site*:

- *bounceInLeft*.
- *bounceInRight*.
- *bounceInDown*.
- *bounceInUp*.
- *zoomIn*.

Finalmente, para incluir os pagamentos via *Paypal* foi preciso seguir os seguintes passos:

- Fazer *login* no ***Paypal***.
- Clicar em ***Merchant Services*** -> ***Key Features*** -> ***Buy Now Buttons***.
- Clicar em ***Step 1 : Choose a button type and enter your payment details***.
- Selecionar ***Buy Now*** na caixa de lista suspense ***Choose a Button Type***.
- Escrever o nome do item no campo ***Item Name***.
- Introduzir o preço do item no campo ***Price*** e selecionar Euros no campo ***Currency***.
- Clicar em ***Step 3: Customize advanced features (optional)***.
- Marcar a *checkbox* ***Take customers to this URL when they finish checkout***.
- Inserir o endereço de retorno desejado após alguém ter completado um pagamento, por exemplo http://grape.pt/2014/8939711559-2.
- Clicar em ***Create Button*** após inserção dos detalhes de pagamento para gerar o código do botão.
- Copiar e colar o código gerado para a página onde se pretende incluir o botão.

Como o aspeto visual do *site* do ENEB entretanto foi alterado pelo cliente, incluem-se de seguida imagens da sua página principal no sentido de refletir o estado no momento de entrega. Todos os *sites* se encontram *online* e podem ser acedidos através dos endereços indicados no início deste anexo.

Página Principal do *Site* do ENEB

Figura 27 – Cabeçalho, *slider* e secção de novidades

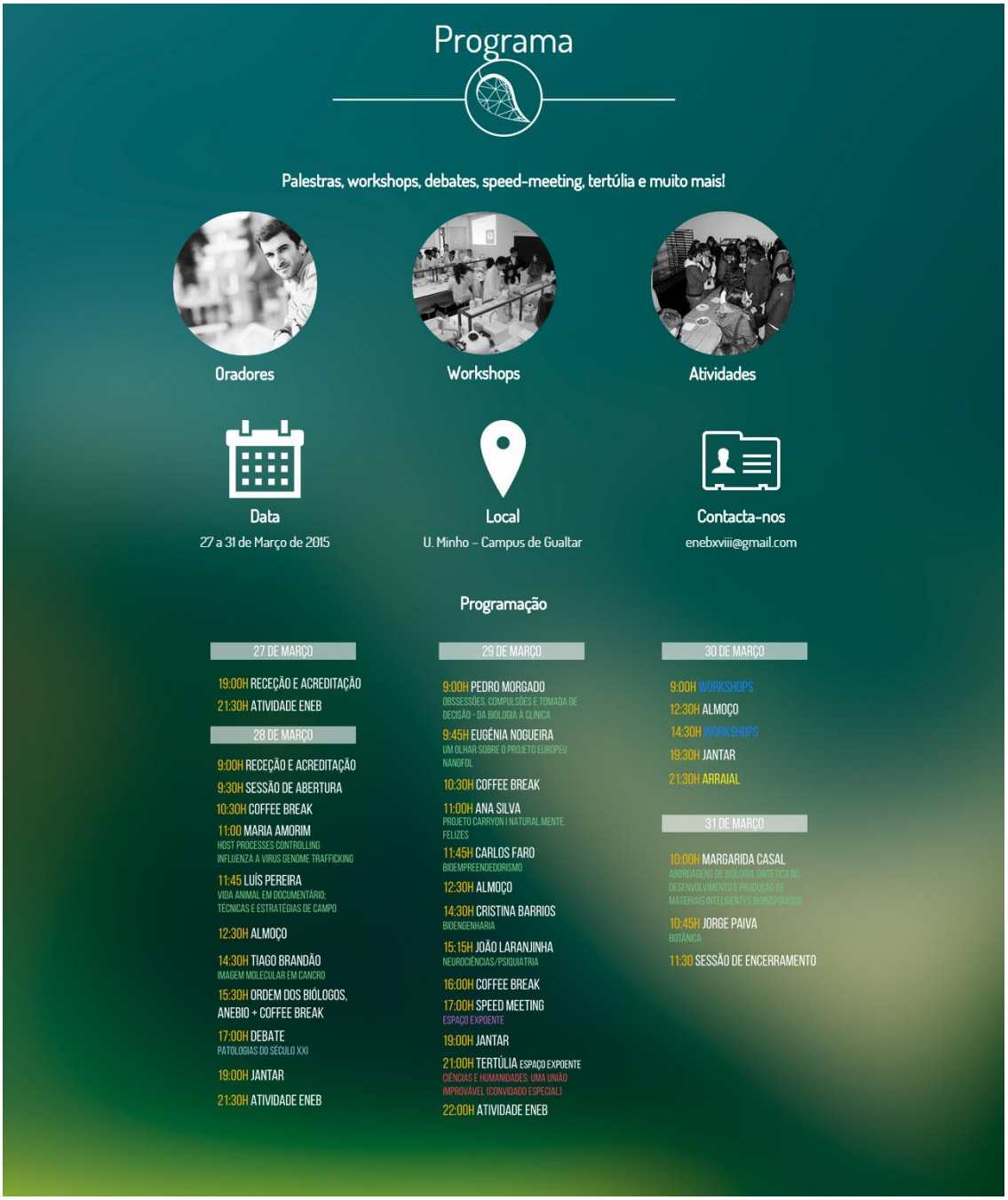

Figura 28 – Programa do evento

Figura 29 – Descrição do evento

Como Chegar?

Porque nem todos os caminhos vão dar a Braga...

A partir do Aeroporto Sá Carneiro

O aeroporto mais próximo da cidade de Braga é o aeroporto Sá Carneiro, situado no Porto, a cerca de 50 km de distância.

A partir deste ponto, e para além do serviço de táxi, no caso de optar pela viagem de metro para se movimentar do Aeroporto para o centro da cidade do Porto poderá usar a Linha E: Aeroporto – Estádio do Dragão, linha violeta.

A paragem de saída será Campanhã, que dá acesso directo à Estação de Caminhos-de-Ferro da Campanhã. A partir desta Estação partem comboios com destino à Estação de Caminhos-de-Ferro de Braga.

Por outro lado, poderá usufruir de autocarros da empresa "getBUS" que fazem ligações diretas entre o Aeroporto Sá Carneiro e a Central de Camionagem de Braga.

Aeroporto Sá Carneiro
http://www.ana.pt/pt-PT/Aeroportos/Porto/Porto/Paginas/HomePorto.aspx
+ 351 229 432 400

Metro do Porto
http://www.metrodoporto.pt/
+351 225 081 000

De Autocarro

No caso de optar pela deslocação de autocarro até à Universidade do Minho, em Braga, poderá consultar aqui algumas das companhias que garantem as viagens, em função do local de partida:

Rede Expressos
Horários disponíveis em: www.rede-expressos.pt
+351 213 581 460

Transdev
Horários disponíveis em: www.transdev.pt
+351 213 581 460

Arriva
Horários disponíveis em: www.transpor.pt

Rodonorte
Horários disponíveis em: www.rodonorte.pt
+351 259 340 710

getBUS
Horários disponíveis em: http://www.getbus.eu/
+351 253 262 371

Figura 30 – Instruções de como chegar ao local

Onde Ficar?

Pavilhão

Como nos anos anteriores, sendo já uma das tradições do ENEB, não poderia faltar o centro de convívio e descanso: o pavilhão. Será utilizado um pavilhão escolar que reunirá todas as condições necessárias à estadia de todos os participantes que optem por uma escolha mais económica. O pavilhão possui uma capacidade para a estadia de cerca de 400 pessoas e para sua higiene pessoal.

Para mais informações contacta a organização do evento!

Hotel Meliá (5*)

Situado a poucos minutos do centro da cidade, da Universidade do Minho, do Centro Ibérico de Nanotecnologia e dos destacados centros comerciais presentes na cidade. O Meliá Braga, que é um luxuoso Hotel contemporâneo e de design sofisticado, é perfeito para descobrir e conhecer Braga, graças à sua excelente localização, à qualidade de seus serviços e a suas completas instalações.

www.melia.com/pt/hoteis/portugal/braga/melia-braga/index.html
+351 253 144 000
melia.braga@meliaportugal.com

Hotel Lamaçães (3*)

O Hotel Lamaçães localiza-se próximo da Universidade do Minho, do Centro Ibérico de Nanotecnologia e a 5 minutos de carro do centro histórico da cidade de Braga.
Esta unidade hoteleira oferece todo o conforto e comodidade dos seus 52 quartos, dos quais 8 com cama de casal e 44 quartos com 2 camas, sendo 8 destes com capacidade até 4 pessoas. Todos os quartos estão equipados com casa de banho privativa, telefone, ar-condicionado, televisão, rádio e wireless gratuito.

www.hotel-lamacaes.com/pt
+351 253 603 680
info@hotel-lamacaes.com

Pousada da Juventude (2*)

Situada a sensivelmente 15 min a pé da Universidade, a pousada é uma alternativa aos participantes que pretendem estar instalados mais perto do centro histórico da cidade. Assim como, ter uma estadia mais cómoda a preços simbólicos.

www.micrositesjuventude.gov.pt/Portal/pt/PBraga.html
+351 253 263 279
braga@movijovem.pt

Note-se que nenhum dos estabelecimentos hoteleiros é assegurado pela organização. Nem existe qualquer tipo de parceria com os mesmos

Figura 31 – Sugestões de alojamento

Figura 32 – Secção de apoios e organização, rodapé

Finalizada a descrição técnica dos *sites* em *WordPress* falta apenas detalhar o processo de importação de catálogos, o que irá ser feito no anexo seguinte.

Importação de Catálogos de Produtos

Foram realizadas 22 importações de catálogos, alguns dos quais com cerca de 1 milhão de artigos, para a plataforma de encomendas *online* da *LabOrders*. O fornecedor envia um ficheiro .xlsx (*Excel*) com a informação que tem disponível sobre os seus produtos. Esse ficheiro de catálogo é então transformado em dois ficheiros de importação (referentes à informação dos produtos e preços) mais um ficheiro de erros (que relata eventuais falhas ocorridas durante o processo de transformação) no formato .csv (*Comma-Separated Values*, em Português valores separado por vírgulas). Isto é feito manualmente, nos raros casos em que é possível, ou automaticamente (através de código), de maneira a ter o formato adequado para ser interpretado pelo algoritmo de importação de produtos da plataforma de encomendas. O processo referido requer, por vezes, o *download* das imagens e páginas dos produtos a partir dos *sites* dos fabricantes, com o objetivo de obter informação adicional ou em falta sobre os mesmos, e a definição programática de *cookies*. Também costuma implicar a conversão de unidades comerciais e nomes dos fabricantes para coincidirem com o formato existente na base de dados.

As importações são compostas por um controlador, um modelo e uma vista, sendo que existem duas classes, *Base_Controller* e *Base_Model*, que possuem os atributos e métodos comuns à maioria dos controladores e modelos, respetivamente. Há também uma classe auxiliar, *LabOrders_Matrix*, que corresponde à matriz onde os dados dos produtos são colocados até serem incluídos no ficheiro .csv de importação gerado, e os modelos do produto e da unidade de armazenamento em *stock* (em Inglês *Stock Keeping Unit* ou SKU). Devido à restrição do número de páginas do documento é impossível incluir todas as importações no mesmo e, por esse motivo, apenas será apresentado um exemplo dos novos ficheiros de importação desenvolvidos.

Seguidamente apresenta-se o modelo da importação referente ao fornecedor VWR, cujo catálogo mais recente compreende 900430 produtos:

```
<?php defined('SYSPATH') OR die('No direct access allowed');

class vwr_8_IGC_2015_03_17_Model extends Base_Model {
    public $c   = array( // Columns
        'BRAND'             => 0,
        'REFERENCE'         => 1,
        'SKU_REFERENCE'     => 2,
        'NAME'              => 3,
        'price'             => 4,
        'COMMERCIALUNIT'    => 9,
        'STORAGE'           => 12,
        'CATEGORY'          => 13,
        'SKU_URL'           => 14,
        // NEW COLUMNS
        'TAX'               => 15,
        'IMAGE'             => 16,
        'ID'                => 17,
        'FAMILYID'          => 18,
    );
```

```
    public $A_additional_priceset_columns = array
    (
        'price'    => 19,
    );
    public $A_pricesets_columns_translation = array
    (
        'price'    => 'VWR International - Material de Laboratório, Lda;IGC
Price Set',
    );

    public $base_dir = 'data/vendors/VWR/VWR_8_IGC_2015_03_17/'; // In relation
to Kohana's index.php

    public $input_filename = '';
    public $output_dir = 'bulkimport_csv/';

    public $product_type                    =
Product_Model::INS_VENDOR_NOTVERYGOOD;
    public $sku_type                        = Sku_Model::INS_VENDOR_CATALOG;

    public function load_input_and_output($strip_first_row = TRUE) {
        $start = time();
        iecho('Start checking "' . __FUNCTION__ . '()"...');

        // Loads input data into the matrix
        // Initializes the 4 variables of the "Matrix" model: input, data,
output and errors (COLUMN)
        $matrix = new LabOrders_Matrix();
        $matrix->init($this->base_dir . $this->input_filename, $this->c);

        if($strip_first_row) {
            $matrix->strip_first_row('data');
        }

        iecho('Done! Took: <b>' . (time() - $start) . '</b> seconds.<br/>');

        return $matrix;
    }

    public function get_product_info($A_input_data, $id, $product_keys) {
        $A_product    = array();
        $A_errors     = array();

        // Product related info
        $A_product[$product_keys['id']] = $id;
        $A_product[$product_keys['family_id']] = $A_input_data[$this-
>c['FAMILYID']];
        $A_product[$product_keys['type']] = $this->product_type;
        $A_product[$product_keys['reference']] = $A_input_data[$this-
>c['REFERENCE']];
        $A_product[$product_keys['name']]   = $A_input_data[$this->c['NAME']];
        $A_product[$product_keys['brand']] = $A_input_data[$this->c['BRAND']];
        $A_product[$product_keys['category']] = !empty($A_input_data[$this-
>c['CATEGORY']]) ? $A_input_data[$this->c['CATEGORY']] : 'undef';
        $A_product[$product_keys['description'] = !empty($A_input_data[$this-
>c['STORAGE']]) ? 'Temperature: ' . $A_input_data[$this->c['STORAGE']] : '';
        $A_product[$product_keys['commercialunit']] = $A_input_data[$this-
>c['COMMERCIALUNIT']];
        $A_product[$product_keys['image']] = $A_input_data[$this->c['IMAGE']];
        $A_product[$product_keys['url']] = '';
```

```
        // Sku related info
        $A_product[$product_keys['vendor']] = 'VWR International - Material de
Laboratório, Lda';
        $A_product[$product_keys['sku_type']] = $this->sku_type;
        $A_product[$product_keys['sku_reference']] = $A_input_data[$this-
>c['SKU_REFERENCE']];
        $A_product[$product_keys['sku_compressedreferences_extra']] = '';
        $A_product[$product_keys['sku_commercialunit']]    = '';
        $A_product[$product_keys['tax']] = 23;
        $A_product[$product_keys['promo']]    = '';
        $A_product[$product_keys['sku_url']] = $A_input_data[$this-
>c['SKU_URL']];

        // Get prices info
        $this->_get_prices_info($A_product, $A_input_data);

        $this->validate_product_info($A_product, $product_keys, $A_errors);

        ksort($A_product);

        return array($A_product, $A_errors);
    }

    /**
     * Removes commercial unit's redundant information
     * Manually builds the sku reference from the web link
     * @param type $matrix
     */
    public function manually_build_commercialunit_and_sku_reference(& $matrix)
{
        $start = time();
        iecho('Start checking "' . __FUNCTION__ . '()"...');

        foreach($matrix->data as $row => $A_row) {
            $commercial_unit = $matrix->data[$row][$this->c['COMMERCIALUNIT']];

            if(strpos($commercial_unit, '1 * ') !== FALSE)
                $matrix->data[$row][$this->c['COMMERCIALUNIT']] =
string::strip_to_plain_utf8(str_replace('1 * ', '',$commercial_unit));

            $sku_url_expression = "@.*?article_number=(.+)@siu";
            $sku_url = $matrix->data[$row][$this->c['SKU_URL']];
            preg_match($sku_url_expression, $sku_url, $A_match);

            if(isset($A_match[1]))
                $matrix->data[$row][$this->c['SKU_REFERENCE']] = $A_match[1];
        }

        iecho('Done! Took: <b>' . (time() - $start) . '</b> seconds.<br/>');
    }

    public function replace_mapped_brands(&$matrix, $col_index) {
        $start = time();
        iecho('Start checking "' . __FUNCTION__ . '()"... ');

        // Mapped brands array filepath
        $filepath = Mapbrand_Model::$brand_base_dir .
Mapbrand_Model::$output_translate_filename;

        // Get the mapped brands array
        $A_mapped_brands = array();
```

```
        // Import previous mapping
        if(file_exists($filepath)) {
            $import_information = file_get_contents($filepath);
            $A_mapped_brands  = json_decode($import_information, true);
        }

        // Fill the new column with the new compressed reference
        foreach($matrix->data as $row => $A_row) {
            $brand_name = $A_row[$col_index];
            $compressed_brand_name =
Product_Model::getCompressedReference($brand_name);

            if(array_key_exists($compressed_brand_name, $A_mapped_brands))
                $matrix->data[$row][$col_index] =
$A_mapped_brands[$compressed_brand_name];
        }

        iecho('Done! Took: <b>' . (time() - $start) . '</b> seconds.<br/>');
    }

    public function replace_default_brand_image_names(& $matrix) {
        $start = time();
        iecho('Start checking "' . __FUNCTION__ . '()"...');

        foreach($matrix->data as $row => $A_row) {
            $brand = $matrix->data[$row][$this->c['BRAND']];

            switch($brand) {
                case 'ABNOVA':
                    $matrix->data[$row][$this->c['IMAGE']] = 'abnova.jpg';
                    break;
                …
                default:
                    $matrix->data[$row][$this->c['IMAGE']] = '';
            }
        }

        iecho('Done! Took: <b>' . (time() - $start) . '</b> seconds.<br/>');
    }
}
?>
```

Código 52 – Modelo da importação do fornecedor VWR

E o conteúdo do controlador respetivo é o seguinte:

```
<?php defined('SYSPATH') OR die('No direct access allowed');

class vwr_8_IGC_2015_03_17_Controller extends Base_Controller {
    protected $memory_limit   = '2048M';

    protected $max_exec_time  = 21600;

    public function __construct() {
        parent::__construct();

        $this->model = new vwr_8_IGC_2015_03_17_Model();
        $this->model_name = get_class($this->model);

        $this->model->input_filename = $this->input->post('filename', '');
    }

    public function build_import_files() {
        ini_set('ini_setmemory_limit', '2048M');

        $start_time = time();

        // Just a shortcut name to use the model column constants
        $m = $this->model;

        // Loads the input and output matrixes, and the errors column
        $matrix = $m->load_input_and_output();

        // Fills an ID column to save the default product ordering (this will
be useful to reorder products after building the family ids)
        $matrix->build_id_column('data', $m->c['ID']);

        // Removes commercial unit's redundant information (1*)
        // Manually builds the sku reference from the web link
        $m->manually_build_commercialunit_and_sku_reference($matrix);
        $m->replace_default_brand_image_names($matrix);

        // Checks if all references are unique in the csv file. If there are
repeated references only the first valid row will be considered
        $m->remove_repeated_references_from_data($matrix);
        // Removes all non-numeric characters
        $matrix->parse_price_col('data', $m->c['price'], $m-
>input_decimal_separator);

        // Replaces the brands with the ones in the Laborders mapped brands
file
        $m->replace_mapped_brands($matrix, $m->c['BRAND']);

        // Sorts by brand and then name. This will preserve array keys
        $matrix->sort_by_multiple_cols('data', $m->c['BRAND'], 'ASC', $m-
>c['NAME'], 'ASC');

        // Tries to determine the Family ID
        $m->build_familyid_col($matrix);

        // Rearrange the products to default ordering
        $matrix->sort_by_col('data', $m->c['ID'], 'ASC');

        $this->process_output($matrix);

        $end_time = time();
```

```
        echo '<p><b>Code ended!</b> (took ' . (($end_time - $start_time) / 60) .
' minutes)</p>';
    }
}
?>
```

Código 53 – Controlador da importação do fornecedor VWR

Apresenta-se na **Figura 32** o formulário da vista de importação automática.

Figura 33 – Formulário da vista de importação automática

Devido à dimensão do ficheiro original este foi dividido em 19 ficheiros de aproximadamente 50000 produtos cada um. O utilizador seleciona na caixa de lista suspensa a parte do ficheiro que pretende importar e por fim clica no botão ***1. Build Import Files*** para dar início ao processo de geração dos ficheiros de importação.

Respeitando este processo a empresa encontra-se preparada para realizar a importação de quaisquer catálogos de produtos em tempo útil, independentemente da sua dimensão.