UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

PLATAFORMA DE EXECUÇÃO DE TESTES DE ACEITAÇÃO VIA INTERFACE DO UTILIZADOR

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

Versão Pública

Hugo Miguel Rodrigues Duarte

Trabalho de Projeto orientado por:
Prof. Doutor Francisco Cipriano da Cunha Martins
e co-orientado por Eng. Fábio Duarte Beirão

2015

Agradecimentos

Começo por agradecer aos professores Francisco Martins e Eduardo Marques por todo o apoio e ajuda que me deram ao longo deste projeto.

Em segundo lugar, agradeço à empresa TFV – Sistemas Informáticos, S.A pela oportunidade proporcionada, pela ajuda e por todo o acolhimento durante projeto.

Agradeço ao Afonso Pereira e á Raquel Carmona, pela paciência, companheirismo e disponibilidade demonstradas ao longo do projeto.

Um grande obrigado à Ana Rita Vieira e ao Gonçalo Silva, pela paciência, ajuda, apoio e pelos grandes momentos passados em todas etapas da minha vida académica.

Agradeço a todo o pessoal da FCT e FCUL, pela ajuda, apoio e acolhimento dado ao longo do percurso académico.

Agradeço à minha família, em especial aos meus pais, que me apoiaram sempre em tudo e ajudaram me a ultrapassar as várias barreiras que foram aparecendo.

Agradeço a todos os que contribuíram para a realização deste projeto.

Por último, agradeço a todos os meus amigos por todo o apoio e pela companhia nos momentos bons e menos bons.

Aos meus pais, família e amigos.

Resumo

Os testes de aceitação são essenciais para empresas de desenvolvimento de *software*, pois verificam se as funcionalidades das aplicações estão de acordo com a especificação funcional. A identificação e correção de problemas em aplicações antes que estas fiquem disponíveis para os utilizadores traduz-se num aumento da qualidade das aplicações e contribuí para a satisfação dos utilizadores e aumento da sua confiança na aplicação como um todo.

Este projeto de Mestrado em Engenharia Informática aborda o desenvolvimento de uma plataforma de execução paralela de testes de aceitação em cada ciclo de Integração Contínua para aplicações *Windows*. Esta ferramenta irá contribuir para a diminuição do tempo de execução dos testes e deverá apresentar uma interface intuitiva para que uma pessoa que não seja programador a consiga usar.

A ferramenta desenvolvida foi aplicada ao *software* TriPoint desenvolvido pela TFV-sistemas informáticos, S.A, a empresa onde decorre este projeto de mestrado. No entanto, a ferramenta pode ser aplicada a outra qualquer aplicação desenvolvendo o adaptador adequado. Atualmente, os testes de aceitação do TriPoint são efetuados pela equipa de *helpdesk* de forma manual, o que implica um gasto considerável na sua realização, tornando-se crucial a existência de uma ferramenta que faça a execução paralela de testes e que seja fácil de usar por utilizadores não especialistas na construção da aplicação.

A concretização deste trabalho resultou numa plataforma de execução paralela de testes que tem uma boa usabilidade, foi bem recebida pelos utilizadores e que reduz imenso o tempo de execução dos testes nas aplicações *Windows*.

Palavras-chave: Teste de aceitação, Execução paralela de testes, Integração continua

Abstract

The acceptance tests are essential for software development companies, because they verify if the application functionalities are in accordance with the functional specification. Identify and correct problems in applications before they become available for users represents an increase in quality of applications and contributes to user satisfaction and confidence in the application as a whole.

This Master's project in Computer Science approaches the development of a parallel execution platform acceptance tests on each continuous integration cycle for *Windows* applications. This tool will help to reduce the test execution time and must present an intuitive interface for a person, which not a programmer, can use.

The developed tool has been applied to the Tripoint software developed by TFV-Sistemas Informáticos, S.A, the company where this master's project, is being developed. However, the tool can be applied to any other application developing the appropriate adapter. Currently, the tripoint acceptance tests are made by helpdesk team manually, which implies a considerable expense in its realization, becoming crucial the existence of a tool that makes the parallel execution of tests and it is easy to use for users not experts in the construction of the application.

The implementation of this work has resulted in a parallel execution platform of tests that has good usability, it was well received by users and which greatly reduces the runtime of tests in *Windows* applications.

Keywords: Acceptance testing, Parallel execution of testing, Continuous Integration

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A TFV-Sistemas Informáticos S.A, é uma empresa de tecnologias de informação, cujo foco é o desenvolvimento de *software* para o setor turístico em particular, o TriPoint, um sistema de gestão de operadores turísticos com todos os canais de venda e negócio integrado numa única aplicação.

Este *software* passa por todas as etapas do negócio, desde o *front-office* até aos relatórios analíticos de *back-office*, possibilitando um total controlo financeiro e comercial.

Neste projeto está-se a desenvolver uma plataforma de execução de testes de aceitação com um foco no *software* TriPoint, que se chama ITR (Integration Test Runnner). Os testes de aceitação interagem com o sistema *UIAutomation*, permite aceder, identificar e manipular os elementos de interface de um *software* Microsoft.

Esta plataforma é essencial para a empresa, pois devido ao facto dos vários clientes do TriPoint estarem permanentemente a pedir novas funcionalidades e por isso saírem novas versões, é necessário haver uma aplicação que execute os vários testes de cada versão de forma rápida. Esses testes são criados por uma aplicação chamada ARITEx realizada por uma colega da FCUL que se encontra a desenvolver o seu projeto em paralelo com este[21].

Resumidamente, o ARITEx é um sistema de testes de aceitação automatizados, através da simulação da interação humana com a interface de utilizador do TriPoint. Este sistema permite guardar as ações e asserções realizadas na interface do TriPoint e converte-las para um teste, depois esse teste é executado na plataforma de execução de testes de aceitação.

1.1 Motivação

Nos últimos anos, tem havido uma evolução na área de testes de *software*, materializada através de diversas plataformas e investigação na área. Os testes de *software* estão divididos em quatro níveis:

- **Testes unitários** que têm como objetivo provocar falhas causadas por defeitos no funcionamento e na implementação em cada módulo, isoladamente [2].
- **Testes de integração** que têm como propósito provocar falhas associadas às interfaces entre os módulos quando esses são integrados para construir a estrutura do *software* que foi determinada na fase de projeto [2].
- **Testes de sistema** que avaliam o *software* à procura de falhas na sua utilização, como se fosse um utilizador final [2].
- **Testes de aceitação** que são realizados geralmente por um grupo restrito de utilizadores finais do sistema. Esses simulam operações de rotina do sistema de modo a verificar se o seu comportamento está de acordo com o solicitado [2].

Este projeto vai focar-se nos testes de aceitação que são os últimos testes a serem feitos antes de lançar a versão final para o utilizador, como se pode ver na Figura 1.1.

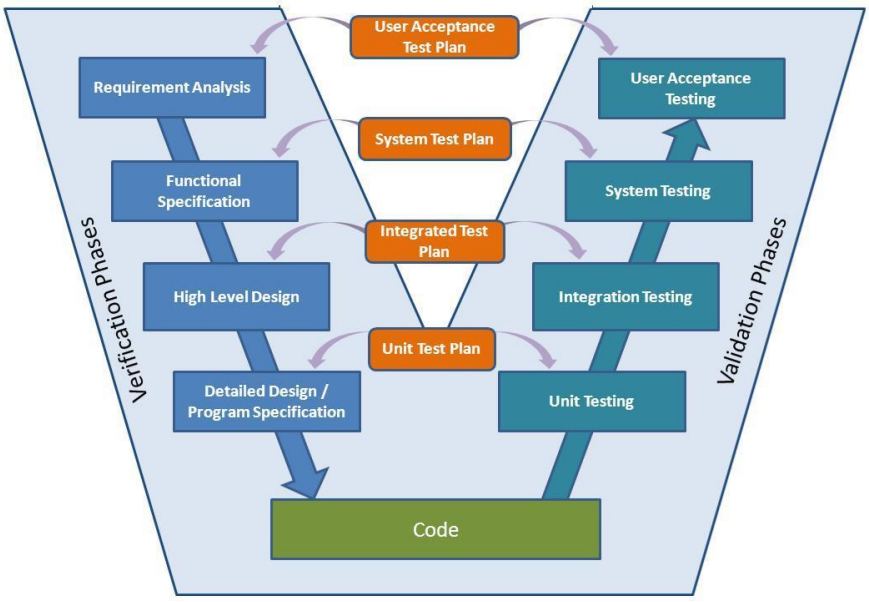

Figura 1.1 - Paralelismo entre as atividades de desenvolvimento e testes de *software*

Os testes de aceitação são muito úteis quando se trata de verificar se as funcionalidades de um sistema estão-se comportando corretamente sem que se tenha de testar manualmente a plataforma, como por exemplo se for feita uma nova versão da plataforma verificar se as funcionalidades novas e antigas ficaram a funcionar bem.

Para isso, já existem algumas plataformas que são mencionadas no Capítulo 2, estas plataformas permitem simular as ações do utilizador, fazer os respetivos testes à aplicação e executa-los. Mas existe algumas funcionalidades que estão em falta nessas aplicações, por exemplo os testes não podem ser executados em paralelo, demorando por isso demasiado tempo a executar. Não há uma análise detalhada dos testes realizados e os utilizadores têm que ser programadores pois as aplicações são muito complexas.

Assim, neste projeto pretende-se desenvolver uma plataforma de execução de testes de aceitação de forma automática que resolva estes problemas.

No contexto da empresa a motivação é reduzir o tempo da execução dos testes de cada versão, reduzir o trabalho das pessoas do *helpdesk*[1] para que tenham mais tempo para fazer outras tarefas sem ser realizar testes ao TriPoint e aumentar o nível de satisfação dos clientes do TriPoint pois vão ser lançadas novas versões com novas funcionalidades mais rapidamente.

1.2 Objetivos

O objetivo do trabalho foi desenvolver uma plataforma de execução paralela de testes de aceitação de forma automática em cada ciclo do sistema de Integração Contínua, garantindo que os desenvolvimentos integrados numa versão não quebram as funcionalidades testadas. Os testes serão gerados a partir de uma especificação precisa das ações a levar a cabo pelo utilizador e dos resultados que são pretendidos. Para concretizar este objetivo, foram definidos quatro componentes do ITR:

1. Uma interface *web* que receba uma bateria de testes focando-se nos testes gerados a partir do Aritex onde o utilizador possa fazer o desenho do grafo de testes, isto é, definir a sequência de execução dos testes tendo em conta as suas dependências. As dependências são os testes executados anteriormente ao teste que se pretende executar, com o objetivo de deixar a aplicação num estado onde seja possível executar o teste. Além disso, esta

[1] Equipa que realiza os testes do TriPoint e dá suporte aos clientes.

interface também tem que fazer a gestão dos utilizadores e dos agentes, o agendamento dos grafos, a gestão das configurações da aplicação (TriPoint) que se pretende testar, a visualização dos resultados dos cenários e uma visualização do que está acontecer nos agentes.

2. Um agente onde o objetivo é ser instalado em vários computadores, ligar-se ao servidor e receber os grafos para executar. Assim, é possível estar a executar em paralelo vários grafos ao mesmo tempo e reduzir o tempo dos testes.
3. Um servidor que faça a gestão dos grafos agendados, isto é, fornecer aos agentes os grafos a executar e receber do componente web os grafos agendados.
4. *Web services* que forneçam os vários serviços / conteúdos da plataforma de execução paralela de testes. Estes serviços vão ser usados pela componente web e pela componente agente.

Além disso, este sistema tem que ser fácil de usar para que uma pessoa que não seja programador consiga usá-lo.

1.3 Contribuições

As principais contribuições desta dissertação são:

- Uma plataforma de execução paralela de testes de aceitação de forma automática para aplicações Windows;
- Desenho e implementação do sistema ITR, compreendendo:
  - Um *website* que faz toda a gestão da plataforma de execução paralela de testes de aceitação;
  - Uma aplicação agente que faz a execução dos vários testes em paralelo;
  - Uma API REST que disponibiliza todos os serviços da plataforma;
  - Integração com o Aritex;
  - Avaliação da ferramenta;
  - Esta plataforma vai reduzir o tempo da execução dos testes de aceitação nas aplicações Windows;

1.4 Estrutura do documento

Este documento está organizado da seguinte forma:

- Capítulo 2: Trabalho relacionado – faz referência a ferramentas já existentes relacionadas com o âmbito da tese.
- Capítulo 3: ITR (Integration Test Runner) – apresenta a aplicação ITR, os conceitos fundamentais subjacentes e a sua funcionalidade geral.
- Capítulo 4: A análise e desenho da solução – apresenta a análise e desenho da solução que consiste na especificação do sistema, definição dos requisitos funcionais e dos requisitos não funcionais, casos de uso e arquitetura.
- Capítulo 5: Implementação – descreve o processo de desenvolvimento, as tecnologias usadas e os diferentes componentes que compõe o sistema.
- Capítulo 6: Avaliação – apresenta uma avaliação da ferramenta, em termos de testes realizados e resultados obtidos, incluindo resultados empíricos da utilização da ferramenta por operadores da TFV.
- Capítulo 7: Conclusão e trabalho futuro – apresenta as conclusões e perspetivas futuras do trabalho desenvolvido.

Capítulo 2

Trabalho relacionado

Neste capítulo apresentamos as ferramentas mais relevantes na área de testes de aceitação. Cada ferramenta será apresentada com uma breve descrição do seu objetivo e das suas funcionalidades. Além disso, é comparada com a ferramenta que desenvolvemos e explicamos o porquê dessa não satisfazer os requisitos funcionais que nos foram apresentados.

2.1 JUnit

O JUnit [4] é uma plataforma de código fonte aberto que permite a realização de testes unitários de aplicações escritas na linguagem de programação Java. Atualmente foi adaptado para outras linguagens, tais como, C# (NUnit), Python (PyUnit), Fortran e C++. Esta plataforma é utilizada por várias ferramentas de teste (algumas delas vão ser analisadas nas secções seguintes), sendo uma das plataformas mais utilizadas nesta área.

Vantagens da ferramenta:

- Facilita a escrita de testes automatizados;
- Integra com as principais IDE's;
- Possui uma grande comunidade de utilizadores;
- Depois de serem escritos, os testes são executados rapidamente sem que seja necessário interromper o processo de desenvolvimento;
- Permite criar uma hierarquia de conjuntos de testes que se aplica a todo ou parte do sistema.

Funcionalidades do JUnit incluem:

- Asserções para testar resultados esperados;
- *Fixtures* (conjunto de dados de teste e objetos utilizados na execução de um ou mais testes) para a reutilização de dados para teste;
- *Test Suites* para organizar e executar conjuntos de testes;

- Interface gráfica e textual para execução de testes;
- Execução de testes em paralelo (a partir da versão JUnit 4.6).

Esta ferramenta tem várias aspetos importantes que vão ser usados na plataforma que pretendemos desenvolver, em particular o paralelismo e parte da interface gráfica. Por outro lado apresenta algumas desvantagens em relação ao que pretendemos, como por exemplo, a execução dos testes em paralelo tem de ser efetuada por programadores, pois só quem realizou os testes é que conhece bem as dependências entre esses testes, caso contrário pode haver conflitos e dar resultados incoerentes. Para além disso, a interface gráfica é muito simples e permite uma análise pouco detalhada sobre os resultados da execução dos testes, só é percetível a programadores que têm acesso ao código fonte testado.

2.2 FitNesse

A ferramenta FitNesse [5] é um *wiki*[2] *web server* que permite efetuar testes a qualquer aplicação onde se testa as camadas de negócio da aplicação. O FitNesse não é uma ferramenta de testes unitários, nem é possível automatizar testes.

Os testes do FitNesse são testes de aceitação que vão integrar as várias camadas da aplicação em conjunto e demonstrar aos programadores e não programadores que a aplicação funciona conforme a especificação funcional. Estes são criados através de tabelas, onde são introduzidos no FitNesse os valores de entrada e os valores esperados e este automaticamente faz a verificação de resultados. Isto é, faz diversas asserções à aplicação que se está a testar.

O FitNesse executa os testes de forma assíncrona, e no final da execução dos testes, as linhas que estiverem a verde passaram no teste e as que estiverem a vermelho não foram bem-sucedidas.

A Figura 2.1 inclui um exemplo de um teste para verificar se uma calculadora está funcionar corretamente em relação à operação divisão.

[2] Wiki é uma aplicação web que permite ao utilizador adicionar, alterar e remover conteúdos em colaboração com os outros utilizadores.

eg.Division		
numerador	denominador	quociente?
10	2	5
12.6	3	4.2
22	7	~ = 3,14
9	3	<5
11	2	4 <_ <6
100	4	33

Figura 2.1 – Especificação do teste a uma calculadora à operação divisão

O FitNesse é uma ferramenta muito simples de testes de aceitação, onde não é possível correr testes em paralelo, é feita uma análise simples dos testes.

Além disso, quere-se incluir no nosso projeto dois aspetos importante do FitNesse. A primeira razão é conseguir testar qualquer aplicação *Windows*. A segunda é ter uma interface *web* que permite ao utilizador executar os testes de qualquer computador.

2.3 Selenium

Selenium [6] é uma ferramenta para se fazer testes de aceitação em aplicações *web* usando o *browser* de forma automatizada. Os testes executam diretamente no *browser*, simulando as ações de utilizadores reais que depois vão ser testadas com asserções.

O Selenium é utilizado essencialmente em duas tarefas do processo de teste:

- Teste de funcionalidade da aplicação *web*;
- Teste de compatibilidade entre navegadores e plataformas diferentes.

As componentes fundamentais desta ferramenta são: o Selenium RC e o Selenium IDE.. O Selenium IDE é uma extensão do navegador Firefox que serve para gravar as ações do utilizador, essas ações podem ser transformadas em código fonte de várias linguagens, depois basta adicionar as asserções desejadas para o teste. O Selenium RC é um servidor desenvolvido em Java que recebe os testes e traduz os comandos Selenium em ações compreendidas pelos *browsers*.

Na Figura 2.2 encontra-se um exemplo de teste para verificar se a calculadora está a funcionar corretamente em relação à operação de percentagem [7].

```
public class rcdemo {
   public static void main(String[] args) throws InterruptedException {

      // Instatiate the RC Server
      Selenium selenium = new DefaultSelenium("localhost", 4444 , "firefox", "http://www.c
      selenium.start();   // Start
      selenium.open("/");  // Open the URL
      selenium.windowMaximize();

      // Click on Link Math Calculator
      selenium.click("xpath=.//*[@id='menu']/div[3]/a");
      Thread.sleep(2500); // Wait for page load

      // Click on Link Percent Calculator
      selenium.click("xpath=.//*[@id='menu']/div[4]/div[3]/a");
      Thread.sleep(4000); // Wait for page load

      // Focus on text Box
      selenium.focus("name=cpar1");
      // enter a value in Text box 1
      selenium.type("css=input[name=\"cpar1\"]", "10");

      // enter a value in Text box 2
      selenium.focus("name=cpar2");
      selenium.type("css=input[name=\"cpar2\"]", "50");

      // Click Calculate button
      selenium.click("xpath=.//*[@id='content']/table/tbody/tr/td[2]/input");

      // verify if the result is 5
      String result = selenium.getText(".//*[@id='content']/p[2]");

      if (result == "5"){
         System.out.println("Pass");
      }
      else{
         System.out.println("Fail");
      }
   }
}
```

Figura 2.2 – Teste Selenium

Em relação à plataforma que pretende-se desenvolver, a parte do Selenium IDE não é muito útil, pois a nossa ferramenta foca-se nas aplicações .NET. No entanto, numa versão mais avançada da plataforma pode ser que haja uma integração dos testes a aplicações *web*. A parte do Selenium RC é relevante para nós porque o funcionamento da nossa aplicação é semelhante, em vez do Selenium IDE temos o Aritex que faz a especificação dos testes e em vez do Selenium RC temos o ITR que traduz os testes do Aritex para ações compreendidas pelas aplicações *Windows*. Além disso, faz ligação com o JUnit que podia ser útil para executar testes em paralelo.

2.4 Protractor

Protractor [9] é uma plataforma de teste ponta-a-ponta para aplicações AngularJS, executa os testes num *browser* real, simulando as ações do utilizador.

O Protractor foi desenvolvido em NodeJS[3] que utiliza por defeito o Jasmine. O Jasmine [10] é uma plataforma BDD[4] para testar código JavaScript, não depende de qualquer plataforma JavaScript e têm uma sintaxe simples e clara que torna fácil a escrita de testes. Para além do Jasmine, também suporta as plataformas Mocha e Cucumber.

Analisando a Figura 2.3, podemos ver que esta plataforma executa em cima do Selenium, pois foi desenvolvida em volta do WebDriverJS e o Selinium Server. Por isso tem todos os benefícios e vantagens dessa ferramenta. Adicionalmente, fornece novos recursos para ajudar a fazer testes de aceitação no AngularJS. Além disso, com o Protractor também é possível usar alguns drivers que implementam WebDriver's wire protocol como o ChromeDriver, onde é possível executar testes sem o Selenium Server.

As principais vantagens desta ferramenta é conseguir fazer teste a múltiplos *websites* ao mesmo tempo, ter uma sintaxe simples e conseguir testar a nossa própria aplicação que é desenvolvida em AngularJS.

A principal desvantagem é não conseguir testar aplicações Windows.

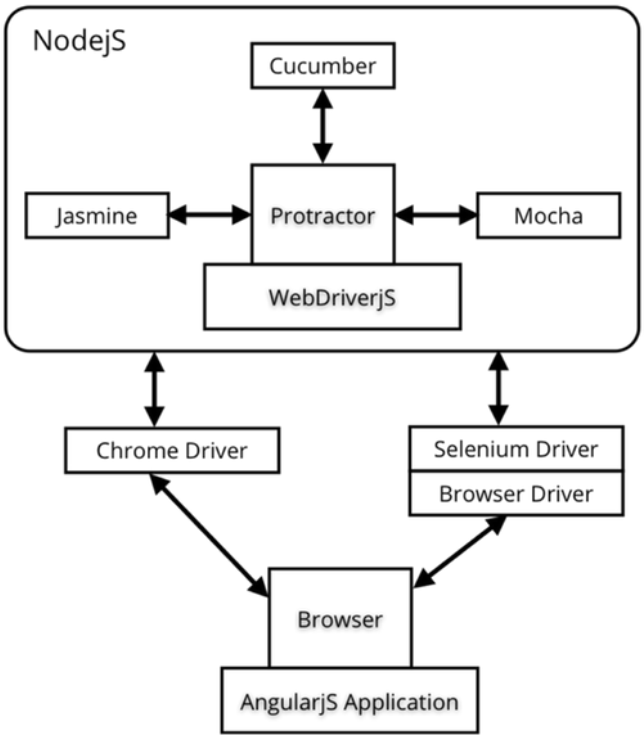

Figura 2.3 – Arquitetura Protractor

[3] Node.js é um interpretador de código JavaScript que funciona do lado do servidor. Onde o objetivo é ajudar programadores na criação de aplicações de alta escalabilidade, com códigos capazes de manipular dezenas de milhares de conexões simultâneas, numa única máquina física.

[4] BDD (Desenvolvimento Guiado por comportamento) é uma técnica de desenvolvimento Ágil que encoraja colaboração entre programadores, setores de qualidade e pessoas não-técnicas ou de negócios num projeto de *software.*

2.5 TeamCity

TeamCity [11] é um servidor de gestão distribuída de *builds* e integração contínua que pode ser instalado em Windows, Linux e plataformas Mac OS.

Esta ferramenta é realizada em torno dos conceitos de um servidor de *build*, uma fila de *build* e uma série de agentes de *build*. Além disso, têm os *Triggers* que vão acrescentar os *builds* pendentes na fila (exemplos de *triggers* incluem *commits* no sistema de controle de versão ou um deadline ser alcançado). O servidor seleciona os agentes inativos para executar os *builds* e organiza os conforme a sua prioridade. Os resultados de cada *build* são mostrados pelo servidor ao utilizador.

Funcionalidades TeamCity:

- Integração Continua e *Testing* – quando é submetida uma versão nova há um conjunto de testes que são realizados e só se houver sucesso em todos os testes é que a nova versão é lançada. Caso contrário, as falhas são imediatamente comunicadas e ficam visíveis para toda a equipa de desenvolvimento;
- Gestão da Construção Efetiva – Definir e executar diferentes tipos de *build* com diferentes configurações de *build* para qualquer projeto, personalizar *build triggers* e fazer uma gestão flexível dos *builds*;
- Manutenção da qualidade do código – é feita uma análise do código por parte do servidor;
- Integração com as principais IDE's;
- Interface *web*.

Examinando as características do TeamCity pode-se observar que há aspetos muito importantes que vamos utilizar na nossa ferramenta, tal como a utilização de um servidor que organiza os agentes para executar os testes de forma paralela. Além disso, vai-se também utilizar uma interface *web* e é uma ferramenta virada para a plataforma .Net.

A principal razão pela qual falamos do TeamCity é porque a empresa onde está a ser desenvolvida o projeto usa esta ferramenta e a ideia era fazer um *plugin* para Team City, e assim, depois de cada *build* da aplicação Tripoint são executados os testes para verificar se as funcionalidades novas e antigas ficaram a funcionar bem.

Capítulo 3

Executor de testes de integração

A aplicação que nos propomos desenvolver é uma plataforma de execução paralela de testes de aceitação, que tem duas propostas base: (a) diminuir o tempo de execução dos testes de aceitação e (b) oferecer uma interface intuitiva para que uma pessoa que não seja programador a consiga usar. Este capítulo apresenta alguns conceitos que são essenciais para entender a aplicação, e descreve o funcionamento das suas interfaces (interface *web* e agente).

3.1 Conceitos

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial expõem-se os conceitos de *passos do teste*, *precedências*, *teste*, *cenário*, *configuração*, *agente*, *execução do cenário* e *utilizador do website*, que serão necessários ao longo da tese.

3.2 Interface *web* da aplicação

Esta secção foi omitida devido à natureza confidencial deste projeto. A versão confidencial expõem as várias vistas da interface *web* da aplicação, explica qual o objetivo de cada uma e o seu funcionamento.

3.3 Interface agente

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial é exposta a interface agente e a sua funcionalidade geral.

Capítulo 4

Análise e desenho da solução

Este capítulo descreve a análise e desenho da solução que propusemos, que consiste na especificação do sistema, definição dos requisitos funcionais e dos requisitos não funcionais, casos de uso e arquitetura.

4.1 Especificação do sistema

A especificação do sistema é uma parte essencial da definição do sistema e proporciona um meio de comunicação com a Empresa. Para a elaboração desta especificação foram realizadas diversas reuniões iniciais onde foram descritas as várias componentes do sistema sem grandes contornos técnicos.

A restante secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial, a mesma expõem os vários componentes da aplicação e o objetivo de cada um.

4.2 Requisitos funcionais

Os requisitos funcionais representam a especificação das funcionalidades e tarefas das componentes da aplicação. Uma má especificação dos requisitos funcionais pode ter um impacto negativo no projeto.

A restante secção foi omitida devido à natureza confidencial deste projeto. A versão confidencial expõem a especificação dos requisitos das componentes da aplicação.

4.3 Requisitos não-funcionais

Os requisitos não funcionais descrevem caraterísticas da aplicação desejadas pelo utilizador, mas que não representam funcionalidades da aplicação.

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial expõem-se os requisitos não funcionais da aplicação.

4.4 Casos de uso

O diagrama de casos de uso mostra de uma forma simples e intuitiva, as funcionalidades da aplicação e a interação dessas funcionalidades com os atores do sistema.

A restante secção foi omitida devido à natureza confidencial deste projeto. A versão confidencial expõem-se o diagrama de casos de uso e a sua especificação.

4.5 Arquitetura

Esta secção descreve a arquitetura do sistema e da aplicação. A arquitetura do sistema mostra uma visão global das várias componentes e qual a ligação que existe entre eles. A arquitetura da aplicação ilustra a estrutura dos pacotes da aplicação implementados.

4.5.1 Arquitetura física do Sistema

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial descreve-se a arquitetura do sistema.

4.5.2 Arquitetura da Aplicação

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial descreve-se a arquitetura da aplicação.

Capítulo 5

Implementação

Nesta fase é descrito o processo de desenvolvimento das várias camadas da ferramenta referidas na arquitetura da aplicação e é feita uma apresentação das tecnologias utilizadas. Para o desenvolvimento destas camadas foi utilizado o ambiente de desenvolvimento *Microsoft Visual Studio*, é um pacote de programas da Microsoft para o desenvolvimento de *software* especialmente dedicado à plataforma .Net. Além disso, é também um bom ambiente para desenvolvimento *web*.

5.1 Tecnologias utilizadas

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial é feita uma apresentação das tecnologias utilizadas, quais as suas vantagens, a razão para a sua utilização e ainda onde se insere cada tecnologia neste projeto.

5.2 Camada de dados

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial expõem-se o modelo relacional da base de dados.

5.3 Camada de negócio

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial expõem-se a camada de negócio da aplicação e os algoritmos utilizados na mesma.

5.4 Camada de serviços

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial expõem-se os vários serviços da API da aplicação.

5.5 Camada de apresentação

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial expõem-se como foi implementado o *front-end* da aplicação.

Capítulo 6

Avaliação

Neste capítulo é feita a avaliação da aplicação ITR, isto é, quais os testes realizados para garantir a qualidade do *software*. Esta avaliação divide-se em três partes: testes de sistema, de carga e de usabilidade.

6.1 Testes de sistema

Os testes de sistema são realizados no final do projeto para garantir que o sistema funciona corretamente como um todo. O principal objetivo é testar todas as funcionalidades na procura de anomalias ou pequenos erros e identifica-los para correção futura.

A restante secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial, a mesma expõem a validação do ITR face aos requisitos funcionais pedidos pela empresa.

6.2 Testes de carga

Após a implementação de todo o sistema e realizados os vários testes de sistema foram realizados os testes de carga. Este tipo de testes tem como principal objetivo verificar se a aplicação possui um bom tempo de resposta tendo em conta o número de testes.

A restante secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial expõem-se os resultados dos testes de carga e a sua análise.

6.3 Testes de usabilidade

Os testes de usabilidade têm como objetivo averiguar a facilidade de utilização e compreensão da aplicação quando manipulada pelo utilizador.

A restante secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial expõem-se os resultados dos testes de usabilidade e a sua análise.

6.3.1 Avaliação da usabilidade

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial é descrita a metodologia usada nos testes de usabilidade.

6.3.2 Avaliação do nível de sucesso na realização das tarefas

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial é descrita a metodologia utilizada na avaliação do nível de sucesso na realização das tarefas.

6.3.3 Resultados

Esta secção foi omitida devido à natureza confidencial deste projeto. Na versão confidencial expõem-se e discute-se os resultados obtidos nos testes de usabilidade.

Capítulo 7

Conclusão e trabalho futuro

Este capítulo apresenta as conclusões finais e as melhorias que poderão ser feitas num trabalho futuro.

7.1 Conclusão

Este projeto apresentou o ITR, uma aplicação de execução paralela de testes de aceitação de forma automática em cada ciclo do sistema de Integração Contínua, garantindo que os desenvolvimentos integrados numa versão não quebram as funcionalidades testadas em outras versões. O ITR tem como objetivo ajudar a empresas a reduzir o tempo da execução dos testes de cada versão da aplicação, e assim, aumentar a satisfação dos clientes, pois vão sair novas versões com novas funcionalidades em menos tempo e com melhor qualidade. Além disso, o ITR pode testar qualquer aplicação Windows.

Atualmente existem diversas plataformas que executam testes de aceitação, mas cujo foco é a construção e a execução assíncrono dos testes, havendo algumas como o JUnit que já faz a execução paralela de testes. Sendo que essa execução paralela tem que ser realizada por programadores pois só o programador que realizou o teste é que conhece as dependências entre testes. Também existe o Selenium e o Protractor que são aplicações onde se pode fazer testes em múltiplas aplicações web, não se aplica a aplicações Windows, mas no futuro pode ser interessante integrar com o ITR para se conseguir testar a própria aplicação e outras aplicações web. Além disso, todas essas plataformas têm as suas interfaces muito simples ou pouco intuitivas, onde um especialista no domínio vai encontrar diversas dificuldades em realizar os testes.

Para a resolução desses problemas foi desenvolvida o ITR que tem uma interface simples, intuitiva e fácil de usar que pode ser utilizada por um programador ou por especialista no domínio. O utilizador tem disponíveis os testes criados onde pode ver os vários passos correspondentes ao teste, e com esses testes criar um cenário de execução paralela. Caso o utilizador não conheça as dependências dos testes, a aplicação ajuda-o

na construção do cenário. Para além disso, é possível correr vários cenários ao mesmo tempo através dos agentes e ver os seus resultados na aplicação.

Na avaliação foram realizados vários testes: validação dos casos de uso, execução de vários cenários de testes sobre o TriPoint para avaliar o bom tempo de resposta tendo em conta o número de testes, e ainda testes de usabilidades com os potenciais utilizadores finais. Estes testes permitiram verificar que o ITR cumpre o propósito para o qual foi concebida, tem uma boa usabilidade, foi bem recebida pelos utilizadores e vai reduzir imenso o tempo de execução dos testes. Também foi possível verificar que todos os requisitos da aplicação estão a funcionar corretamente.

Em termos de desenvolvimento do projeto foram cumpridos os objetivos definidos inicialmente.

7.2 Trabalho futuro

Relativamente ao trabalho futuro, existem quatro pontos na aplicação ITR que precisam de ser melhorados nas próximas versões.

O primeiro ponto é a integração da reposição da base de dados da aplicação TriPoint no ITR. Este ponto é o mais importante e que se tem que fazer o mais rapidamente possível, pois caso não se faça essa reposição da base de dados a um estado inicial cada vez que se corre um cenário tem-se que repor as alterações feitas manualmente, caso contrário quando se executar o cenário outra vez podem ocorrer erros nos testes por causa da replicação dos dados na aplicação TriPoint. Para resolver este ponto é preciso desenvolver um sistema de *snapshots* à base dados que repõe o estado inicial. Devido a existirem várias bases de dados é um desenvolvimento complicado que ainda demora cerca de 2 semanas.

O segundo ponto é melhorar o processo de integração contínua que atualmente é feito manualmente, isto é, no final de cada *build* da aplicação a testar é necessário ir ao ITR e correr os respetivos cenários. Para resolver esta situação é necessário desenvolver um *plugin* para o TeamCity para que no final de cada *build* da aplicação sejam executados os cenários do ITR automaticamente. Este desenvolvimento é capaz de demorar cerca de 1 mês.

O terceiro ponto é alterar a aplicação agente de modo a ser um serviço como o servidor, em vez de ser uma aplicação WinForms. Devido ao facto das informações mostradas pela aplicação WinForms serem desnecessárias, pois essas informações são possíveis de visualizar no *website* que é a única funcionalidade dessa aplicação para além da autenticação. Além disso, os agentes já não têm autenticação nem são adicionados pelo administrador. Para resolver esse problema no momento da instalação

do serviço, o agente é adicionado à base de dados onde o nome do computador onde está a ser instalado vai ser a sua identificação. Cada vez que o serviço do agente executa é criada uma ligação ao servidor, como já está a ser feita de momento só que com a aplicação WinForms. Esta implementação é fácil e deve demorar cerca de 1 semana a ser desenvolvida.

Por último, é fazer a implementação das melhorias sugeridas pela equipa de *helpdesk* enumeradas na secção 6.3.3 que devem demorar 1 ou 2 semanas a desenvolver, pois são tarefas simples.

Bibliografia

[1] TFV Sistemas informáticos s.a, 2004. [Online]. Available: http://www.tfv.pt. [Acedido em 25 11 2014].

[2] A. R. C. Rocha, J. C. Maldonado e K. C. Weber, Qualidade de Software - Teoria e prática, Prentice Hall, 2001.

[3] R. D. Craig e S. Jaskiel, Systematic Software Testing, Artech House Publishers, 2002.

[4] E. Gamma e K. Beck, "JUnit," JUnit, 2002. [Online]. Available: http://junit.org/. [Acedido em 24 11 2014].

[5] G. Adzic, Test Driven .Net Development with FitNesse, Paperback , 2008.

[6] D. Burns, Selenium 1.0 Testing Tools: Beginner's Guide, Olton: Packt Publishing, 2010.

[7] TutorialsPoint, "Selenium – Remote Control" [Online]. Available: http://www.tutorialspoint.com/selenium/selenium_rc.htm. [Acedido em 29 06 2015].

[8] J. Huggins, "Selenium" ThoughtWorks, 2004. [Online]. Available: http://www.seleniumhq.org/. [Acedido em 24 11 2014].

[9] "Protractor" [Online]. Available: https://angular.github.io/protractor/#/. [Acedido em Junho 2015].

[10] "Jasmine" [Online]. Available: http://jasmine.github.io/edge/introduction.html. [Acedido em Junho 2015].

[11] V. Melymuka, TeamCity 7 Continuous Integration Essentials, Birmingham: Packt Publishing, 2012.

[12] RFC 1321, section 3.4, "Step 4. Process Message in 16-Word Blocks".

[13] Microsoft, "Fundamentos do Entity Framework 4" [Online]. Available: https://msdn.microsoft.com/pt-br/library/jj128157.aspx. [Acedido em 29 06 2015].

[14] Code Project, “Introduction to ASP.NET Web API” [Online]. Available: http://www.codeproject.com/Articles/549152/Introduction-to-ASP-NET-Web-API. [Acedido em 29 06 2015].

[15] W3C, “Cascading Style Sheets” [Online]. Available: http://www.w3.org/Style/CSS/. [Acedido em 29 06 2015].

[16] R. R. e. J. ErichGamma, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994.

[17] Microsoft, “Introduction to SignalR” [Online]. Available: http://www.asp.net/signalr/overview/getting-started/introduction-to-signalr. [Acedido em 29 06 2015].

[18] Microsoft, “UI Automation Overview” [Online]. Available: https://msdn.microsoft.com/en-us/library/ms747327(v=vs.110).aspx. [Acedido em 29 06 2015].

[19] Code Project, “UI Automation Overview” [Online]. Available: http://www.codeproject.com/Articles/289028/White-An-UI-Automation-tool-for-windows-application. [Acedido em 29 06 2015].

[20] Quartz.Net, “Quartz Enterprise Scheduler .Net” [Online]. Available: http://www.quartz-scheduler.net/. [Acedido em 29 06 2015].

[21] R. Carmona, “Plataforma de especificação de testes de aceitação via interface do utilizador,” Faculdade de Ciências da Universidade de Lisboa, 2015 (Em publicação).

Anexos

Anexo A

Este anexo foi omitido devido à natureza confidencial deste projeto. Na versão confidencial são descritos os casos de uso da aplicação.

Anexo B

Este anexo foi omitido devido à natureza confidencial deste projeto. Na versão confidencial expõem-se o diagrama de classes da ferramenta desenvolvida.

Anexo C

Este anexo foi omitido devido à natureza confidencial deste projeto. Na versão confidencial expõem-se os testes de sistema realizados à aplicação.

Anexo D

Este anexo foi omitido devido à natureza confidencial deste projeto. Na versão confidencial expõem-se o questionário de usabilidade utilizado para os respetivos testes.