Universidade do Minho
Escola de Engenharia

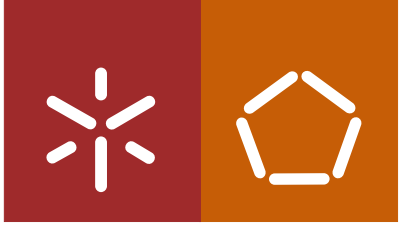
Sérgio Agostinho Machado Pinheiro

Estudo e implementação de testes de software em desenvolvimento ágil

outubro de 2015

Sérgio Agostinho Machado Pinheiro **Estudo e implementação de testes de software em desenvolvimento ágil**

UMinho|2015



Universidade do Minho
Escola de Engenharia

Sérgio Agostinho Machado Pinheiro

Estudo e implementação de testes de software em desenvolvimento ágil

Dissertação de Mestrado
Mestrado em Engenharia de Sistemas

Trabalho efetuado sob a orientação da
Professora Doutora Maria Teresa Torres Monteiro

outubro de 2015

DECLARAÇÃO

Nome: Sérgio Agostinho Machado Pinheiro

Endereço eletrónico: sergiomap8@gmail.com Telefone: 912108764

Bilhete de Identidade/Cartão do Cidadão: 13954838

Título da dissertação: Estudo e implementação de testes de software em desenvolvimento ágil

Orientador/a/es:

Professora Doutora Maria Teresa Torres Monteiro

Ano de conclusão: 2015

Mestrado em Engenharia de Sistemas

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA DISSERTAÇÃO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE.

Universidade do Minho, 28/10/2015

Assinatura: Sérgio Agostinho Machado Pinheiro

Agradecimentos

Gostaria de agradecer em primeiro lugar à *F3M Information Systems, SA* pela possibilidade de realizar esta dissertação e por me ter proporcionado um ótimo ambiente de trabalho. Agradeço particularmente ao Manuel Pereira pela disponibilidade e apoio prestados. A toda a equipa, em especial ao departamento de desenvolvimento, pela receptividade e boa disposição que sempre demonstrou.

À Professora Doutora Teresa Monteiro, minha orientadora, pela ajuda e acompanhamento.

Aos meus pais um enorme agradecimento por todo o suporte, carinho e incentivo. Sem eles todo este percurso académico não seria possível.

À Andreia pelo carinho, companhia e encorajamento.

A todos os meus amigos que me proporcionaram bons momentos e me acompanharam nesta fase importante, muito obrigado.

Resumo

Nesta dissertação é apresentado o estudo e implementação de testes de *software* em desenvolvimento ágil.

Os testes de *software* têm cada vez mais importância para as empresas que desenvolvem *software*, devido à natural evolução das exigências do cliente. Face à necessidade de cumprir as expetativas do cliente, a *F3M Information Systems, SA* sentiu que devia melhorar as suas práticas de testes.

Com base na metodologia de desenvolvimento de *software* Scrum, foi realizada uma análise a três processos de testes identificados pelo International Software Testing Qualifications Board (ISTQB) como orientados a este tipo de desenvolvimento: Test-Driven Development (TDD), Acceptance Test-Driven Development (ATDD) e Behavior-Driven Development (BDD). A análise e comparação dos três processos ditou que o BDD era que se adequava à empresa, pois tem um grande enfoque no cliente e no comportamento final do *software*.

O BDD foi implementado, de forma parcial, num dos projetos mais recentes da empresa desenvolvido em ASP.NET com arquitetura MVC. O processo revelou-se bastante efetivo sobretudo por permitir uma redução significativa do tempo despendido nos testes, devido à sua automatização, e por facilitar a interação entre todas as partes envolvidas desde o início do projeto.

Abstract

In this dissertation the analysis and implementation of software testing in an agile development methodology is presented. Software testing is an area of growing interest for companies that develop software, mainly due to the evolution of the customers requirements. In order to accomplish customers expectations, F3M Information Systems, SA decided to reinforce its testing practices.

Based on the software development methodology Scrum, three processes of software testing identified by the International Software Testing Qualifications Board (ISTQB) were analyzed: Test-Driven Development (TDD), Acceptance Test-Driven Development (ATDD) and Behavior-Driven Development (BDD). According to the analysis and comparison of these processes, BDD was the one that fitted the needs of the company, since it has a big focus on the client and in the final behavior of the software.

BDD was partially implemented in one of the most recent projects of the company, developed in ASP.NET with MVC architecture. The process was proved to be very effective, not only enabling a substantial reduction of the time spent in testing, due to its automation, but also by facilitating the interaction between the stakeholders from the beginning of the project.

Conteúdo

Lista de Figuras

Listagens

Lista de Tabelas

Glossário

build

versão compilada de um *software*.

feature

característica de um *software*.

open source

programa de computador com código de livre acesso ao público.

outside-in

abordagem de fora para dentro, do utilizador para a implementação.

release

lançamento de um produto de *software*.

scrum

framework de desenvolvimento ágil para planeamento e gestão de projetos de *software*.

sprint

iteração do *Scrum*.

stakeholder

parte interessada no negócio.

tester

pessoa que testa *software*.

user story

descrição de uma necessidade do utilizador.

Acrónimos

ATDD	Acceptance Test-Driven Development.
BDD	Behavior-Driven Development.
HTML	HyperText Markup Language.
ISTQB	International Software Testing Qualifications Board.
MVC	Model-View-Controller.
TDD	Test-Driven Development.

Capítulo 1

Introdução

1.1 Testes de *software*

Atualmente quase tudo o que rodeia o ser humano está envolto de tecnologia que por sua vez depende de um *software* para funcionar. Quando se pensa em *software* pensa-se em algo que vai ser utilizado quer de forma direta ou indireta, e como tal, pressupõe-se que funcione sem qualquer problema, o que por vezes não acontece. Se isso acontece é porque algo falhou e que o *software* não tem qualidade.

A qualidade é algo que é inerente na realidade da civilização moderna, qualquer que seja a área em causa, e com o passar do tempo à medida que ocorre uma evolução quer tecnológica, cultural ou social, as expectativas vão aumentando. Na área do *software* é difícil garantir a inexistência de falhas. No entanto, o controlo de falhas depende da tolerância que cada empresa dá a esta questão e qual a consequência de uma falha numa aplicação desenvolvida. Nesse sentido, a garantia de qualidade é a forma de prevenção de erros ou defeitos e de evitar problemas nos produtos. Esta aplica-se também ao *software* procurando verificar se as características e funcionalidades satisfazem as expectativas do cliente.

Os testes de *software* fazem parte de um processo de garantia de qualidade em que é possível medir a qualidade do *software* em termos de de-

feitos relativamente a requisitos funcionais e não funcionais. Envolve a execução de componentes do *software* ou do sistema para avaliar o cumprimento dos requisitos por onde se guia o desenvolvimento, a resposta a diferentes tipos de entradas, o tempo de resposta a certos pedidos, a sua usabilidade, o comportamento em diferentes ambientes, entre outras propriedades.

O propósito dos testes de *software* é transversal a qualquer tipo de ambiente ou metodologia utilizada. No entanto, a metodologia utilizada no desenvolvimento do *software* irá determinar quando e como vão ser efetuados os testes, uma vez que cada metodologia tem diferentes fases e tempos de desenvolvimento. Os objetivos do produto e a importância que uma empresa dá ao cliente também influenciam o processo de testes, dado que terá de haver um foco em alguns níveis ou tipo de testes em detrimento de outros.

Esta dissertação que vai incidir sobre testes de *software* vai ser desenvolvida na empresa *F3M Information Systems, SA*, uma das maiores empresas portuguesas especializadas em Tecnologias da Informação e Comunicação. O trabalho consiste no estudo e implementação de um processo de testes de *software* em ambiente de desenvolvimento ágil que foi proposto pela empresa, uma vez que consideram que atualmente é uma área onde a abordagem utilizada não é a correta.

1.2 Motivação e Objetivos

A maioria das empresas de desenvolvimento de *software* deixa o tema da qualidade de lado ou não lhe dá a devida relevância. O controlo da qualidade é um processo por vezes demorado e acarreta custos que as empresas entendem que não compensam. Acaba por ser uma atitude errada pois o custo evitado nesse controlo pode refletir-se depois quer na reparação desses erros, quer na opinião do cliente relativamente ao produto.

Apesar da resistência por parte das empresas relativamente a este

tema, tem cada vez mais importância garantir que o *software* desenvolvido tem qualidade, pois o cliente está cada vez mais exigente.

A *F3M Information Systems, SA* também tem lacunas nesse aspeto, devido à falta de um processo de testes de *software*. A prática utilizada para testar o *software* baseia-se apenas na revisão das funcionalidades após a sua implementação sem o critério e a organização necessários, o que resulta num elevado número de falhas nas suas aplicações. Estas falhas podem ter grande impacto principalmente a nível financeiro devido à perda de clientes, reparos, vendas não concretizadas ou até perda no valor da marca.

Uma vez que se trata de uma dissertação de mestrado desenvolvida num ambiente empresarial um dos objetivos é que o trabalho desenvolvido venha a ter utilidade para a empresa. Visto que existe a necessidade de introduzir práticas que garantam a qualidade do *software*, esta dissertação visa estudar e implementar um processo de testes de *software*. Este estudo, que será efetuado tendo em conta que são utilizadas metodologias de desenvolvimento ágeis, terá como base os processos de testes identificados pelo ISTQB para este tipo de metodologias. Ao longo deste estudo, vai-se evitar que o processo escolhido provoque grandes alterações na corrente habitual de desenvolvimento. Será apenas acrescentado um processo de testes que se adeque às necessidades da empresa e que possa melhorar de forma direta a qualidade do *software* desenvolvido.

Assim, as questões de investigação são:

- Quais são os problemas da empresa ao nível dos testes de *software* e quais as suas possíveis soluções?
- Que processos de testes podem ser implementados tendo em conta a metodologia de desenvolvimento utilizada na empresa?
- Quais as ferramentas de testes que podem ser escolhidas, conhecendo as linguagens de programação e o *software* utilizado na empresa, para uma possível integração?

- De que forma pode ser implementado o processo de testes sem que haja alterações nos hábitos de desenvolvimento?
- Pode este processo permitir a cooperação de outras pessoas da equipa durante os testes, para além do *tester*?
- Quais os níveis e os tipos de testes que vão ser utilizados?

1.3 Estrutura da dissertação

Esta dissertação encontra-se dividida em seis capítulos e um anexo. No Capítulo 2 vão ser abordados todos os conceitos essenciais sobre testes de *software*, a sua generalidade e qual a sua importância. Também se vão abordar quais as metodologias de desenvolvimento de *software* existentes, com ênfase na metodologia ágil *scrum*. No Capítulo 3 são descritos os testes em desenvolvimento ágil, desde a sua organização, níveis e uma descrição sucinta dos processos de testes utilizadas em metodologias de desenvolvimento ágil. No Capítulo 4 são apresentados os problemas atuais na empresa relativos à falta de testes. São comparados os processos abordados no capítulo anterior e por fim é descrita a plataforma de desenvolvimento utilizada na empresa e quais as ferramentas a utilizar ao longo deste trabalho. No Capítulo 5 é feita uma breve descrição do projeto sobre o qual vai incidir o processo de testes e toda a implementação do processo escolhido. No Capítulo 6 são apresentadas as conclusões e ideias para o trabalho futuro. No Anexo A consta todo o código da implementação que não foi incluído no Capítulo 5.

Capítulo 2

Testes de Software

Este capítulo retrata aquilo que são os testes de *software* na sua generalidade. Inicialmente vai ser referida a importância dos testes de software e de seguida apresentam-se alguns conceitos essenciais sobre testes e também sobre metodologias de desenvolvimento.

2.1 Porquê testar?

Um erro cometido no desenvolvimento de um software resulta num defeito nesse produto. Quando esse defeito for executado irá resultar em algo inesperado causando assim uma falha. No entanto, nem todos os defeitos são causados por erros no código. Uma das causas comuns de defeitos são as lacunas nos requisitos que podem estar associados a requisitos não funcionais como a testabilidade, escalabilidade, desempenho ou segurança.

A função dos testes passa por medir a qualidade do software em termos de defeitos, tanto relativamente a requisitos e características funcionais como não funcionais, com vista a reduzir a possibilidade de ocorrência de problemas. Para tal deve haver um processo adequado de forma a minimizar a probabilidade de haver problemas, e o nível de risco do sistema. À medida que esse processo é utilizado, em projetos anteriores,

torna-se mais evidente a causa dos defeitos, o que ajuda a melhorar o processo em projetos futuros.

As atividades de testes devem ser integradas num processo de controlo de qualidade, juntamente com boas práticas de desenvolvimento ou análises de defeitos.

Ao optarem por um processo de testes, as empresas garantem que haverá uma redução significativa do número de erros, em relação a um processo em que não se façam testes.

2.2 Modelos de desenvolvimento de software

Os testes são atividades que estão relacionadas com as metodologias de desenvolvimento de *software*. Para cada metodologia de desenvolvimento são necessárias abordagens diferentes para os seus testes.

Dentro das metodologias de desenvolvimento existem as sequenciais e as iterativo-incrementais. Nas metodologias sequenciais o progresso flui pelas diferentes fases do projeto e só se avança para a fase seguinte depois da anterior estar concluída. Exemplos de modelos sequenciais são o modelo *Waterfall* e o Modelo V. O modelo iterativo-incremental é um processo cíclico que surgiu em resposta às fraquezas dos modelos sequenciais. Os modelos de desenvolvimento ágeis, o *Rapid Application Development* (RAD) e o *Rational Unified Process* (RUP) são exemplos de metodologias iterativo-incrementais [MSB11].

O processo de ambas as metodologias de desenvolvimento, sequencial e iterativo-incremental, são representados na figura 2.1.

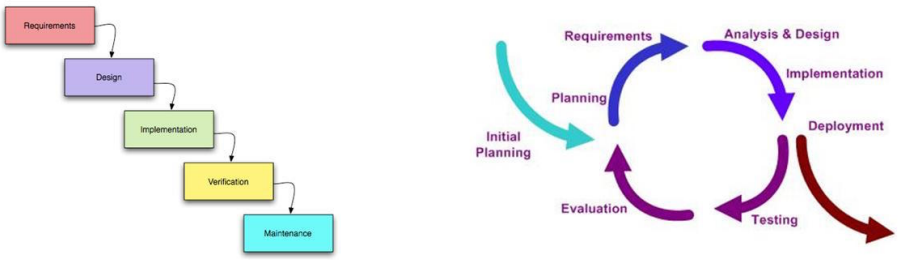


Figura 2.1: Processo de uma metodologia sequencial (*Waterfall*) e uma iterativo-incremental

Como o trabalho desta dissertação vai ser realizado num ambiente onde é utilizada uma abordagem ágil, apenas se vai aprofundar esta metodologia de desenvolvimento.

2.2.1 Modelos Ágeis

O desenvolvimento ágil de *software* é um conjunto de metodologias que surgiu através do *Agile Manifesto* [Bec01] em 2001. Este tipo de desenvolvimento facilita uma comunicação frequente e desde cedo entre todos os intervenientes no projeto, o que permite que se evitem erros desde o início.

O *Agile Manifesto* assenta nos seguintes valores:

- Indivíduos e interações mais do que processos e ferramentas;
- Software funcional mais do que documentação abrangente;
- Colaboração com o cliente mais do que negociação contratual;
- Responder à mudança mais do que seguir um plano.

A relevância dos indivíduos e interações refere a importância da auto-organização e motivação no desenvolvimento ágil. A valorização do *software* funcional em detrimento da documentação é enfatizada, pois dá um rápido *feedback* à equipa de desenvolvimento e terá uma melhor aceitação por parte dos clientes. A colaboração com o cliente permite um

melhor entendimento do que é pretendido e é suscetível de trazer sucesso para o projeto. Como a mudança é algo inevitável nos projetos de software e existem fatores que podem ter grandes influências sobre o projeto e seus objetivos, é importante ter flexibilidade nas praticas de trabalho ao invés de seguir um plano rígido.

2.2.2 Abordagens ágeis

Existem várias abordagens para o desenvolvimento ágil. Cada uma implementa valores e princípios do *Agile Manifesto* de maneira diferente. As abordagens mais utilizadas são o *scrum*, *Extreme Programming* e *Kanban*. Neste trabalho apenas será abordado o *scrum*, uma vez que é a metodologia de desenvolvimento utilizada no ambiente de trabalho.

Scrum

O *scrum* foi definido por Hirotaka Takeuchi e Ikujiro Nonaka em 1986 como "uma estratégia de desenvolvimento onde a equipa trabalha como uma unidade para atingir um objetivo comum" [TN86]. Posteriormente, já no início do século XXI e depois de alguns trabalhos com esta metodologia, Ken Schwaber e Mike Beedle descrevem o método no livro *Agile Software Development with scrum* [SB02].

Scrum é uma *framework* que tem sido utilizada para gerir o desenvolvimento de produtos complexos desde 1990. É uma maneira de as equipas trabalharem de forma unida no desenvolvimento de um produto. Cada equipa tem associados papéis, eventos, artefactos e regras. Estes componentes servem um propósito específico e são essenciais para o uso do *scrum*.

O desenvolvimento ocorre em partes pequenas, chamadas *sprints*, em que cada *sprint* é uma "peça" que encaixa nas que foram criadas anteriormente. Produzindo uma "peça" de cada vez, incentiva a criatividade e permite que as equipas respondam ao *feedback* e mudança, para que seja feito apenas o necessário [Sch04].

Eventos

O principal evento é a *sprint*, que divide um projeto em iterações com um tamanho fixo, geralmente entre duas a quatro semanas. Antes de se iniciar cada *sprint* é realizado o *Product Backlog* e a *sprint Backlog*. No *Product Backlog* é feita a análise de requisitos, que consiste na lista dos itens planeados para o produto - inclui todas as *sprints* - de forma priorizada. O *sprint Backlog* é feito antes do inicio de cada *sprint*, onde são selecionados os itens com maior prioridade do *Product Backlog* para serem realizados no *sprint*.

Todos os dias são feitas reuniões, *daily scrum*, onde a equipa de desenvolvimento reporta e atualiza o estado da *sprint* atual. No final de cada *sprint* é feita a soma dos itens do *Product Backlog* que foram completados durante a *sprint* e todos os anteriores, chamado *Product Increment*. Nesta altura o *Product Increment* deve estar realizado de acordo com a definição de concluído (DoD). Cabe depois ao *Product Owner* decidir se o liberta [SS11].

O processo do *scrum* é ilustrado na Figura 2.2.

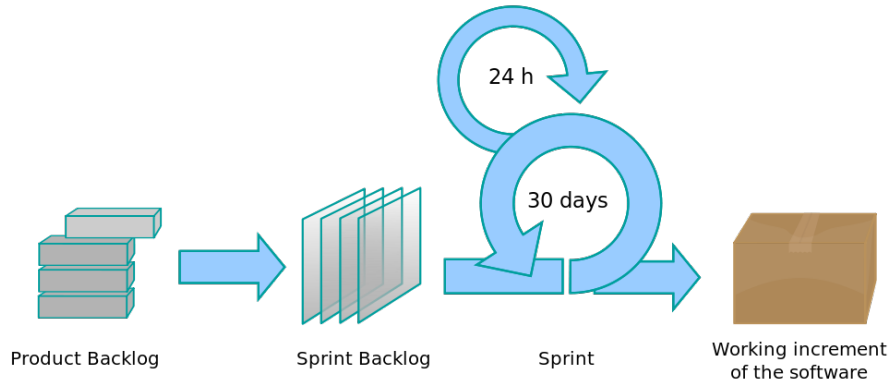


Figura 2.2: Processo do scrum [Lak09]

Funções

Para este processo existem três funções diferentes [SS11]:

- **Scrum Master** - garante que as práticas e regras do *scrum* são

implementadas e resolve problemas a nível de recursos ou outros que possam impedir o bom funcionamento da equipa;

- **Product Owner** - representa o cliente e faz a análise de requisitos;
- **Equipa de desenvolvimento** - desenvolve e testa o produto.

2.3 Testes de software - níveis

Os níveis de teste servem para identificar as áreas em falta e evitar sobreposição e repetição entre as fases do ciclo de vida do desenvolvimento. Cada fase de um processo de desenvolvimento de *software* tem que ser testada. Os vários níveis de teste são ilustrados na Figura 2.3.

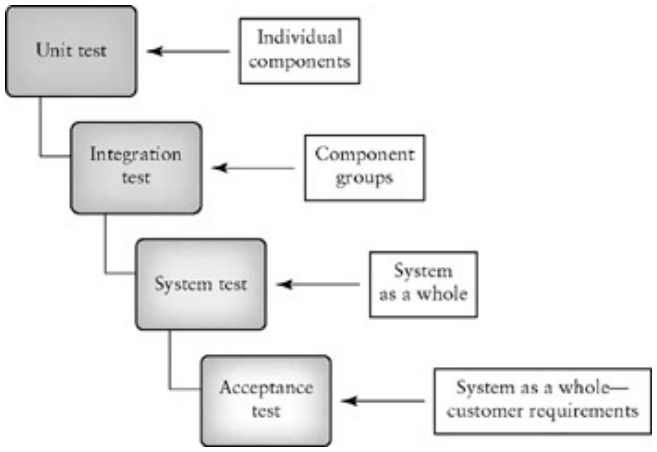


Figura 2.3: Níveis de teste

Os níveis de teste vão ser detalhados em seguida.

2.3.1 Testes unitários

Os testes unitários verificam o funcionamento de módulos de software, programas, objetos ou classes que possam ser testados separadamente. Este tipo de testes são escritos pelos programadores durante o desenvolvimento e consistem no isolamento de partes do programa para evitar

dependências, de forma a que haja a certeza que uma determinada função faz o esperado e para que os erros sejam corrigidos logo que forem detetados [Cop04].

2.3.2 Testes de integração

Os testes de integração procuram testar as interações entre diferentes partes de um sistema. O propósito dos testes de integração é fazer uma verificação funcional, de desempenho e de requisitos de confiabilidade dos componentes. Estes testes são feitos com base nos módulos testados nos testes unitários agrupando-os em componentes resultando assim num sistema integrado.

Este tipo de testes exige o conhecimento da arquitetura em causa para que se perceba a sua influência no processo de integração [Cop04].

2.3.3 Testes de Sistema

Os testes de sistema testam o comportamento de todo o sistema, para verificar se este cumpre os requisitos especificados. Focam-se nos defeitos que surgem num alto nível de integração.

Nestes testes o programa deve funcionar conforme o esperado e o ambiente deve ser idêntico ao ambiente do utilizador final de forma a minimizar o risco de falhas específicas devido ao ambiente e que não foram detetadas em testes anteriores [Cop04].

2.3.4 Testes de aceitação

Os testes de aceitação são em alguns casos da responsabilidade do utilizador final. Estes testes servem para verificar se o sistema se comporta de acordo com o que foi pedido e determinam a satisfação do cliente relativamente ao produto [Cop04].

2.4 Testes de software - tipos

Para além dos níveis de teste existem os tipos de testes que são focados num objetivo particular do teste. Dependendo dos seus objetivos os testes são organizados de maneira diferente. De seguida são apresentados os quatro tipos de testes de *software*.

2.4.1 Testes caixa-preta

Os testes de caixa-preta, ou testes funcionais são um tipo de testes baseados nos requisitos funcionais. Nestes testes são testadas as funcionalidades do sistema sem entrar na sua estrutura interna, apenas é considerado o comportamento externo do *software*. Exemplo destes testes são os testes exploratórios [MSB11].

2.4.2 Testes não funcionais

Os testes não funcionais referem-se a aspetos do *software* que não estão relacionados com nenhuma função, como o desempenho ou a escalabilidade. São os testes necessários para medir as características dos sistemas que podem ser quantificadas, como os tempos de resposta nos testes de desempenho [MSB11].

2.4.3 Testes caixa-branca

Os testes caixa-branca ou estruturais avaliam o comportamento interno dos componentes do software. Este tipo de testes atuam diretamente no código para avaliar aspetos como condições, ciclos ou caminhos lógicos. Exemplo deste tipo de testes são os testes unitários [MSB11].

2.4.4 Testes de regressão

Os testes de regressão são a repetição de testes a uma aplicação que sofreu alterações de modo a corrigir defeitos para garantir que essas alterações não tiveram uma implicação negativa na aplicação. Caso estas alterações resultem em novos defeitos considera-se que o sistema regrediu. Este tipo de testes inclui testes funcionais, não funcionais e estruturais [MSB11].

2.5 Testes manuais e automatizados

Testes manuais definem-se como o processo de executar cada tarefa de testes de forma manual e comparar os seus resultados com as expectativas, de modo a encontrar defeitos no programa. Essencialmente, testes manuais são a utilização de um programa como utilizador final, através de vários cenários e garantindo que funciona corretamente. Em projetos de pequena dimensão ou de curto prazo, onde não existe um procedimento rigoroso de testes, os testes exploratórios podem ser suficientes. No caso de grandes projetos terá de ser adotado um procedimento sistematizado. No entanto, os testes manuais estão sempre presentes num projeto pois certas tarefas de testes não dispensam a intuição humana.

Os testes automatizados utilizam ferramentas, que controlam a execução dos testes, baseadas em algoritmos para comparar os resultados obtidos com os previstos. Algumas tarefas de testes podem ser extensas e trabalhosas para se fazerem testes manuais e requerem maior eficácia, daí terem surgido os testes automatizados. Em projetos com dimensões consideráveis e que podem vir a ter muitos utilizadores, este tipo de testes é o mais adequado, uma vez que permitem que os testes sejam executados rápida e repetitivamente.

Face aos testes manuais, os testes automatizados requerem custos superiores na sua fase inicial, ou seja, quando são criados. Em contraste, as execuções dos testes manuais representam um custo muito mais elevado

do que os testes automatizados. A Figura 2.4 mostra esses custos quando os testes são repetidos muitas vezes.

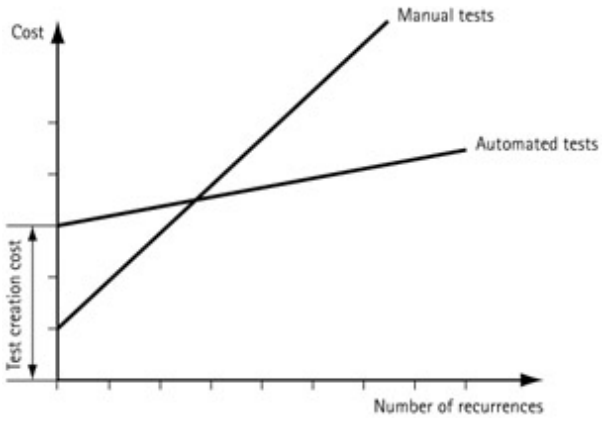


Figura 2.4: Comparação de custos nos testes manuais e automatizados [Lin03]

Capítulo 3

Testes em desenvolvimento ágil

Neste capítulo são abordados os testes em desenvolvimento ágil. Inicialmente será dada uma visão das diferenças entre os testes nas metodologias tradicionais e metodologias ágeis. De seguida, serão abordadas as atividades principais, de um modo geral, para o desenrolar dos testes em desenvolvimento ágil, como o seu planeamento e os níveis de testes. Por fim, vão-se dar a conhecer as técnicas de testes utilizadas em desenvolvimento ágil, com a exposição dos seus propósitos e uma explicação do processo de cada uma.

3.1 Organização dos testes em desenvolvimento ágil

Como foi descrito no Capítulo 2, o desenvolvimento ágil é uma das várias metodologias de desenvolvimento que se podem adotar. Devido às suas características e à sua organização, existem certos procedimentos a ter em conta durante os testes.

No desenvolvimento ágil existe uma grande interação entre os intervenientes em todos os passos do desenvolvimento. Os *testers* trabalham de perto com os programadores e com a equipa de negócio para garantir que são atingidos os níveis de qualidade pretendidos. Esta proximidade

começa na criação de testes de aceitação juntamente com a equipa de negócio e prossegue com os programadores para definir determinadas estratégias.

Comparativamente com as metodologias de desenvolvimento tradicionais, como a *Waterfall*, as tarefas de testes não acontecem apenas no final do processo, antes do lançamento. Nas metodologias ágeis, como são iterativas e incrementais, as atividades de teste ocorrem ao longo da iteração. Estas iterações são altamente dinâmicas, com algum paralelismo e sobreposições. Na Figura 3.1 é possível observar as diferenças de procedimentos.

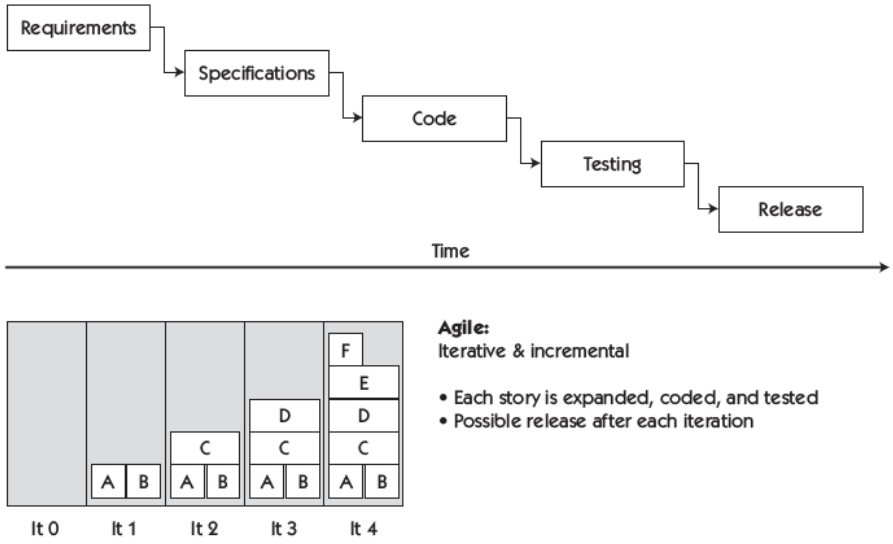


Figura 3.1: Waterfall *vs* Agile [CG09]

Este processo iterativo permite que haja uma comunicação frequente entre a equipa, e consequentemente que seja dado *feedback* atempado sobre a qualidade do produto de modo a que a equipa se possa focar nas funcionalidades mais importantes.

Em cada iteração é necessário juntar todos os componentes do produto já desenvolvidos. Surge aqui a integração contínua que permite que os testes automatizados sejam executados regularmente para que haja uma fusão de todas as alterações feitas no *software* e a integração de

todos os componentes alterados.

No final de cada iteração, as funcionalidades desenvolvidas resultam em software funcional com características com valor para o negócio [CG09].

3.1.1 Planeamento

Ao nível do planeamento pode-se considerar que existem dois tipos, o planeamento da *release* e o planeamento da iteração. O planeamento da *release* pode ocorrer meses antes do início do projeto e visa refinar o *product backlog*. Este inclui atividades como definição de prioridades, analise de risco, a definição dos níveis de testes e quais as *release*s que vão ser testadas. De uma forma geral, este planeamento fornece a base para uma abordagem de alto nível aos testes. O planeamento da iteração acontece depois do planeamento da *release*, de acordo com as prioridades definidas anteriormente. Neste planeamento pretende-se definir as *user stories* e o trabalho necessário por cada uma delas durante a *sprint*. Aqui é feita uma análise mais pormenorizada quanto às funcionalidades que vão ser testadas, tais como o que vai ser testado, como, e qual o esforço necessário para essas tarefas [Coh04].

3.1.2 Níveis de testes

Enquanto nas metodologias de teste sequenciais os níveis de teste dependem uns dos outros, ou seja, o critério de saída de um nível é o critério de entrada no nível seguinte, nas metodologias ágeis isso não acontece. No desenvolvimento ágil, durante uma iteração os níveis de testes sobrepoem-se uns aos outros, devido a alterações no projeto. Cada funcionalidade progride pelas atividades seguintes [CG09]:

- Nível unitário, feito pelo programador
- Testes de aceitação divididos em duas atividades: teste de verificação da funcionalidade, geralmente automatizado; teste de valida-

ção da funcionalidade, geralmente manual e com a colaboração de toda a equipa e equipa de negócio.

Para além disto, durante a iteração ocorrem os testes de regressão onde são executados os testes automatizados da iteração atual e das anteriores, num processo de integração contínua.

Podem também existir em alguns projetos, o nível de sistema que inclui testes funcionais, de performance, usabilidade entre outros.

3.1.3 Configuração

Nas metodologias ágeis recorre-se muito a ferramentas de automatização dos testes. Estas ferramentas são utilizadas pelos programadores para os testes unitários mas também podem ser utilizadas para os testes funcionais num nível de testes mais alto. No entanto, a frequência com que estes testes são executados é diferente, por exemplo, os testes unitários devem ser executados cada vez que é feito *check-in* do projeto, enquanto os testes funcionais basta correrem de vez em quando. Os testes automatizados permitem uma integração continua com o sistema e ajudam a diminuir o risco de regressão associado a alterações frequentes pois podem ser executados repetidamente e sempre que se pretenda [BK02].

3.1.4 Papel de um *tester*

O papel de um *tester* inclui atividades que geram e fornecem *feedback* acerca do estado dos testes, do seu progresso, da qualidade do produto e do processo. Essas atividades incluem [GVVE08]:

- Perceber e implementar a estratégia de testes;
- Medir e reportar a cobertura dos testes;
- Garantir que são utilizadas as ferramentas de testes adequadas;
- Configurar, utilizar e gerir os ambientes de teste e os seus dados;

- Reportar falhas e trabalhar com a equipa para as resolver;
- Treinar outros elementos da equipa em aspetos de teste relevantes;
- Garantir que são agendadas as tarefas de teste adequadas;
- Colaborar ativamente com os programadores e com os *stakeholders* para clarificar requisitos;
- Participar nas reuniões da equipa, sugerir e implementar melhorias.

3.2 Processos de testes

Existe uma série de práticas que podem ser adotadas em qualquer tipo de desenvolvimento de *software*. No desenvolvimento ágil procura-se que essas práticas sejam introduzidas desde cedo para que sejam utilizados os devidos tipos de testes no momento e no nível adequados, para garantir a qualidade no produto.

3.2.1 Test-Driven Development

TDD é uma técnica que consiste no desenvolvimento de testes unitários antes da conceção do código, que virão a servir como um guia durante o desenvolvimento. Inicialmente esta técnica foi descrita como uma prática utilizada em *Extreme Programming*, no entanto, tem sido utilizada com outras metodologias de desenvolvimento nomeadamente as metodologias ágeis.

No TDD, para cada pequena funcionalidade o programador escreve o teste especificando e validando o código do programa que numa fase inicial serve apenas para passar no teste concebido previamente. Este código depois é refeito para aprimorar as funcionalidades do programa [JS05].

O ciclo de desenvolvimento do TDD ilustrado na Figura 3.2, assenta nas seguintes ações [Bec03]:

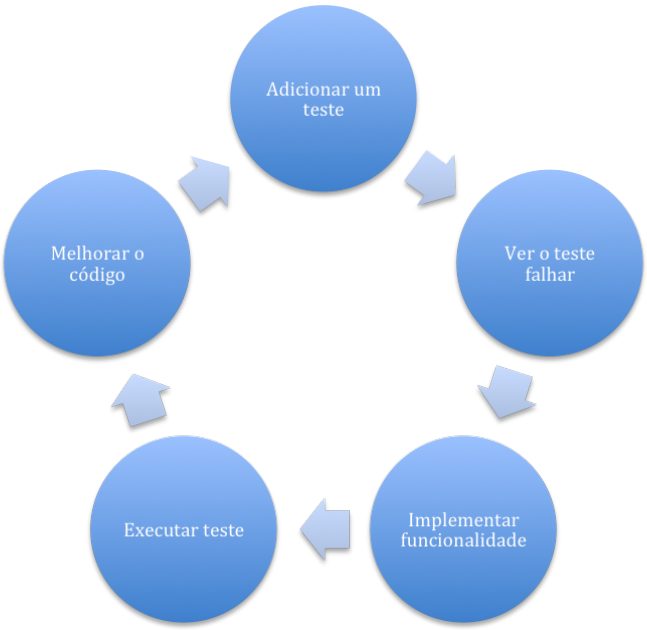


Figura 3.2: Ciclo do TDD

- **Adicionar um teste**. Cada nova funcionalidade é iniciada com um teste que é criado com base nas especificações recolhidas. Este teste, antes da implementação, permite que o programador se foque nos requisitos antes de implementar uma funcionalidade, ao invés dos testes que são feitos depois de uma funcionalidade estar implementada.

- **Ver o teste falhar**. Os testes são executados e vão falhar pois as funcionalidades ainda não foram implementadas. Estes testes servem como validação daquilo que é esperado, ou seja, que o teste irá falhar.

- **Implementar a funcionalidade**. É necessário escrever o código que vai implementar as funcionalidades, no entanto, pretende-se apenas o mínimo para que o teste possa passar. A funcionalidade será melhorada no fim do processo.

- **Executar o teste**. Se os testes tiverem sucesso significa que cumprem os requisitos.

- **Melhorar o código**. À medida que o código vai aumentando deve ser feita uma limpeza com alguma regularidade. Esta limpeza permite que o código se torne mais legível e que a manutenção seja mais fácil, o que será valorizado numa fase posterior do desenvolvimento.

O processo é repetido para cada pequeno pedaço de código.

Os testes feitos são principalmente focados no código, embora no decorrer do processo possam ser escritos nos níveis de integração ou sistema.

3.2.2 Acceptance Test-Driven Development

O ATDD é muito similar ao TDD e surgiu com o propósito de possibilitar uma maior comunicação acerca do que se pretende que o sistema faça. Com o ATDD toda a equipa envolvida num projeto colabora para ganhar clareza e entendimento compartilhado antes do desenvolvimento começar.

Enquanto que no TDD os testes unitários estão relacionados com o código e com o ponto de vista do programador, no ATDD os testes de aceitação dão uma visão externa do sistema e estão voltados para o utilizador.

Os critérios de aceitação e os testes são definidos durante a criação das *user stories*, e estes testes podem ser reutilizados e servem também como testes de regressão.

Existem duas práticas principais no ATDD: antes da implementação são criados exemplos das funcionalidades, depois estes exemplos são transformados em testes de aceitação automatizados [Adz09].

Visão geral do ATDD

O processo do ATDD inicia-se com uma reunião entre a equipa de desenvolvimento, o *tester* e a equipa de negócio. São feitas perguntas para

que possam ser definidos exemplos de *user stories*, de forma a que haja um bom entendimento daquilo que se pretende e para que estes exemplos possam ser escritos como testes.

Após a obtenção das respostas sobre o que se pretende do sistema, o passo seguinte é a criação dos testes de aceitação. Estes testes, feitos numa linguagem acessível a todas as partes envolvidas, descrevem características especificas do que é pretendido e que virão a ajudar a equipa de desenvolvimento a implementar as funcionalidades corretamente. Antes disso estes testes são validados por alguém responsável pela área de negócio. De seguida é feito o código para os testes, sob a forma de testes unitários, num processo igual ao praticado no TDD. Depois de codificados estes testes são executados e são sinalizados como falha, partindo-se assim para a implementação da funcionalidade.

Quando a funcionalidade estiver implementada e os testes forem bem sucedidos, os requisitos são revistos pelo *Product Owner* e outras partes interessadas. Esta revisão pode levar ao aparecimento de novos requisitos ou a alterações nos existentes [Hen08]. O processo descrito é ilustrado na Figura 3.3.

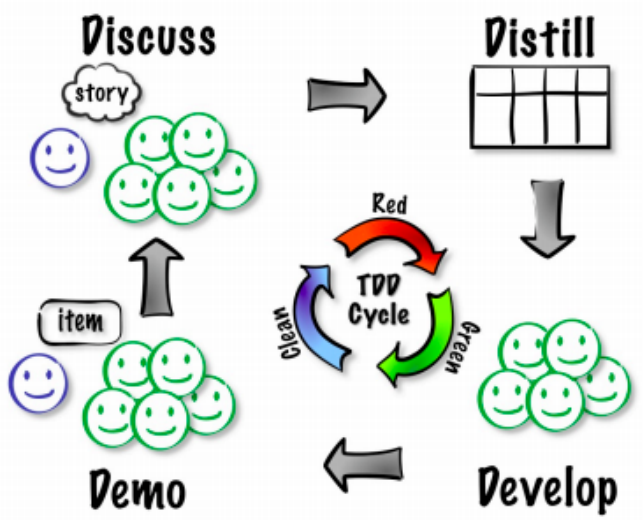


Figura 3.3: Ciclo do ATDD [Hen08]

3.2.3 Behavior-Driven Development

O BDD é um processo que combina técnicas e princípios do TDD. Este processo surgiu com a necessidade de haver maior comunicação entre as partes interessadas num projeto, daí ter uma linguagem ubíqua e de fácil entendimento.

Dan North criou o BDD com o intuito de repensar a forma como são concebidos os testes unitários e de aceitação. Aqui o foco principal é o comportamento em vez da estrutura. Toda a implementação é dirigida com base no comportamento pretendido e são valorizadas as interações entre as pessoas, sistemas e objetos. O BDD surge assim como um veículo de comunicação entre as diferentes funções num projeto de *software*. Para facilitar as interações e a comunicação é utilizada a linguagem ubíqua *Gherkin*. Esta linguagem é partilhada por todas as pessoas envolvidas no projeto, quer tenham conhecimentos técnicos ou não.

Assim, é possível exemplificar o comportamento pretendido para uma aplicação com uma linguagem percetível para os analistas de negócio, *testers* e programadores.

Visão geral do BDD

Figura 3.4: Ciclo do BDD

O ciclo do BDD inicia-se com a definição de um objetivo de negócio. Em reunião, a equipa do projeto juntamente com utilizadores finais e outras partes interessadas no projeto debatem por forma a perceberem quais as funcionalidades que devem ser criadas.

Uma vez que se trata de uma prática que necessita da colaboração e do conhecimento de todas as partes, torna-se mais uma vez necessário que estas trabalhem em conjunto na definição das *features*. As *features* representam, num alto nível, as principais características do sistema. Após a definição das *features*, estas são ilustradas através de exemplos concretos daquilo que se pretende. Estes exemplos utilizam uma linguagem ubíqua já enunciada anteriormente. Por exemplo, um cenário de uma mensagem de boas-vindas após efetuar *login*:

```
Scenario: User is greeted upon login
   Given the user "Aslak" has an account
   When he logs in
   Then he should see "Welcome, Aslak"
```

Estes exemplos constituem a base para a construção do sistema, servindo como critérios de aceitação e como guia para o desenvolvimento. Os exemplos são posteriormente convertidos em especificações executáveis, através de uma ferramenta, cada uma das frases do exemplo são transformadas em métodos. Estes métodos serão codificados para automatizar aquilo que se pretende no respetivo passo.

O BDD também se desenrola num nível mais baixo, através das especificações de baixo nível, o que também ajuda os programadores a escrever código mais confiável, mais fácil de fazer manutenção e melhor documentado. Estas especificações de baixo nível guiam o programador no desenvolvimento das funcionalidades. Estas são similares a testes unitários mas são feitas de forma a transmitir o objetivo do código e fornecer um exemplo de como este deve ser utilizado, ou seja, também são escritas com base no comportamento esperado [Sma14].

3.3 Quadrantes dos testes em metodologias ágeis

Os quadrantes dos testes em metodologias ágeis, definidos por Brian Marick, alinham os níveis de testes com os tipos de testes adequados. O modelo dos quadrantes de testes exibido na Figura 3.5 ajuda a garantir que os tipos e níveis de testes importantes são incluídos no processo de desenvolvimento [CG09].

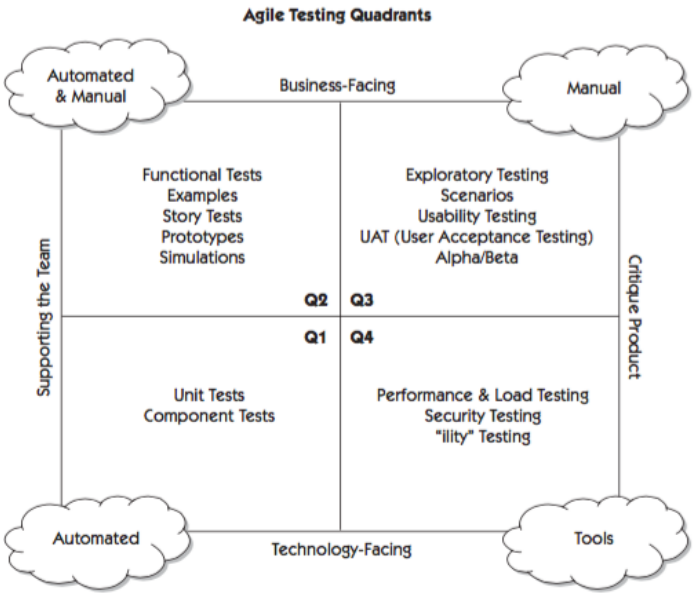


Figura 3.5: Quadrantes de testes das metodologias ágeis [CG09]

A ordem de numeração dos quadrantes não está relacionada com o momento em que cada teste deve ser feito. O momento para cada tipo de teste está relacionado com os objetivos e riscos associados ao projeto.

Quadrante 1

O quadrante 1, Q1, representa os testes num nível mais baixo, o típico TDD. Aqui incluem-se os testes unitários e os testes de componentes, ambos podem ser automatizados com ferramentas da família xUnit.

O objetivo principal deste quadrante é orientar os programadores durante o desenvolvimento, o que lhes permite ter maior confiança na escrita do código.

Quadrante 2

Os testes no quadrante 2, Q2, também ajudam o trabalho da equipa de desenvolvimento, embora que num nível mais alto. Estes testes representam a qualidade desejada pelos clientes, por isso se chamam testes voltados para o negócio ou para o cliente. Estes testes derivam de exemplos debatidos com o cliente que descrevem cada funcionalidade. São escritos numa linguagem que possa ser entendida pela área de negócio.

A maior parte destes testes necessitam de ser automatizados, de forma a fornecer informação de forma rápida e para que possam ser executados com frequência como parte de um processo de integração contínua.

Quadrante 3

Os testes do quadrante 3, Q3, são utilizados para criticar o produto. Este quadrante serve para identificar se o software desenvolvido está dentro das expectativas ou não. Para isso tenta-se simular o comportamento do utilizador final. Em certos casos estes testes podem ser automatizados, mas noutros apenas a intervenção humana poderá ditar a qualidade do produto.

Quadrante 4

O tipo de testes quadrante 4, Q4, é transversal a qualquer metodologia de desenvolvimento. Estes testes têm o objetivo de analisar questões de desempenho, robustez e segurança, através de ferramentas especializadas.

Capítulo 4

Problemas identificados e soluções propostas

Neste capítulo vão ser abordados os problemas da empresa relativos à falta de um processo de testes. Após a exposição desses problemas, vão ser comparados os três processos explicados no capítulo anterior por forma a que se possa decidir qual o que se enquadra melhor para resolver os problemas atuais. Será também abordada a plataforma de desenvolvimento sob a qual vão ser feitos os testes e quais as ferramentas a utilizar no processo de testes.

4.1 Processo de testes atual

Como foi referido no Capítulo 2, a *F3M Information Systems, SA* utiliza o Scrum como metodologia de desenvolvimento, e em cada *sprint* os programadores são alocados a diferentes funcionalidades. Os testes fazem parte das tarefas de determinada funcionalidade, no entanto, estes são realizados de forma *ad hoc*. Após o desenvolvimento de uma funcionalidade o programador verifica se o programa executa e se o que foi desenvolvido está a funcionar conforme o esperado.

4.1.1 Problemas atuais na empresa

Visto que não existe nenhum processo estruturado de testes nem nenhum planeamento, a empresa depara-se com um elevado número de erros. Estes erros advêm maioritariamente do levantamento de requisitos ou do desenvolvimento. Por vezes o levantamento de requisitos não é feito da forma correta ou não é devidamente transmitido a quem desenvolve o produto, e as funcionalidades desejadas pelo cliente não são implementadas ou não funcionam conforme o esperado.

Este problema dos requisitos é transversal a muitas empresas, na maior parte das vezes devido à falta de comunicação entre os *stakeholders*. Quanto aos erros que são consequência do desenvolvimento estes devem-se à falta de conhecimento técnico ou engano na conceção de determinada funcionalidade.

Através das estatísticas apresentadas na Tabela 4.1, relativas aos três semestres anteriores, é evidente o elevado número de erros nos produtos da empresa e o consequente tempo perdido no tratamento desses erros.

Tabela 4.1: Estatísticas dos erros no software da empresa

Período	**1ºS 2014**	**2ºS 2014**	**1ºS 2015**
% tempo gasto na correção de erros	15	14	13
Nº de horas de projeto para gerar erro	32	32	33
Nº de erros	520	495	412

Estas estatísticas são reflexo da falta de um processo rigoroso de testes ao *software* desenvolvido.

Os processos já abordados e que vão ser agora comparados visam reduzir os números apresentados anteriormente e, por conseguinte, aumentar a qualidade do *software*. Através destes processos procura-se também promover um uso mais eficiente dos recursos e do tempo.

4.2 Comparação TDD, ATDD e BDD

Os três processos explicados no Capítulo 3 vão ser agora comparados por forma a escolher apenas um que se enquadre melhor na empresa e naquilo que se pretende obter dos testes.

Todos os processos que foram estudados têm algumas semelhanças. O BDD e o ATDD surgiram a partir do TDD, devido a necessidades relacionadas com uma maior intervenção de todos os *stakeholders* do projeto.

O TDD é um processo muito técnico, com um grande enfoque no código e na estrutura de uma aplicação o que ajuda na orientação dos programadores. Contudo, a experiência do utilizador (UX) não faz parte do processo, ou seja, não é representado o comportamento de um utilizador e o que realmente se espera da aplicação. Outra das limitações do TDD é não existir um meio comum entre os *stakeholders*, o que leva a problemas de comunicação. Por ser muito técnico e não existir uma linguagem comum que torne os testes percetíveis a pessoas que não possuam os conhecimentos técnicos pode causar lacunas em certos aspetos que necessitam da colaboração de diferentes áreas.

Quando se fala em BDD ou ATDD ou Especificações por exemplos, pretende-se um entendimento comum do que está a ser feito.

O BDD é uma prática ágil que permite uma melhor comunicação entre programadores, *testers*, áreas de negócio e pessoas não-técnicas, durante um projeto de *software*, descrevendo um ciclo de iterações com saídas bem definidas e resultando na entrega de *software* testado e que funciona como o esperado. O ATDD é a prática de expressar requisitos funcionais como exemplos concretos antes do desenvolvimento das funcionalidades.

O BDD sendo um processo *outside-in* foca-se no comportamento do sistema, enquanto que o ATDD se foca na recolha dos requisitos nos testes de aceitação que vão guiar o desenvolvimento. Isto faz com que o BDD seja focado unicamente no cliente, enquanto o ATDD se inclina

para o programador, tal como o TDD, sem se saber se o sistema faz o que realmente devia fazer. Assim, o BDD dá uma compreensão mais clara sobre o que o sistema deve fazer a partir da perspetiva do programador e do cliente e revela-se muito mais eficiente de que nos processos tradicionais em que o cliente simplesmente escreve um documento com as funcionalidades que pretende.

Neste caso o processo que se adequa melhor à realidade da empresa é o BDD uma vez que permite o envolvimento de todas as partes durante a definição das funcionalidades, tudo é desenvolvido com base no comportamento esperado, o que por sua vez reflete a preocupação com a usabilidade e com o cliente.

4.3 Plataforma ASP.NET

Parte dos projetos da *F3M Information Systems, SA* são desenvolvidos com a plataforma web *Microsoft ASP.NET*. O ASP.NET é baseado em *.NET Framework* e pode ser escrito em várias linguagens, como *C#, F#* e *Visual Basic* .NET. O ambiente utilizado e mais comum é o *Visual Studio* uma vez que fornece todas as características necessárias para o desenvolvimento em .NET [Mic15a].

O projeto que serve de suporte a esta dissertação é desenvolvido em ASP.NET MVC com *Repository Patterns*.

O padrão de arquitetura Model-View-Controller (MVC), ilustrado na Figura 4.1, *Model-View-Controller*, separa as aplicações web em três camadas:

- Modelo - armazena os dados que são recuperados pelo controlador e apresentados na camada de visualização.
- Visualização - pede informação ao controlador e apresenta-a ao utilizador.
- Controlador - lida com as interações, atualiza o modelo e passa a

informação para a camada de visualização.

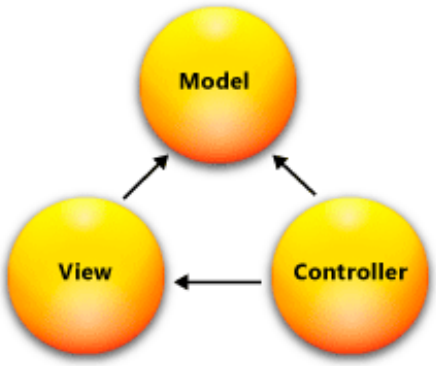


Figura 4.1: Interações no modelo MVC [Mic15b]

Este padrão ajuda na criação de aplicações onde se pretende separar os diferentes aspetos da aplicação. Especifica onde cada tipo de lógica se deve localizar na aplicação. A separação das camadas permite a independência destas, e também que o foco da implementação seja apenas num aspeto de cada vez [Mic15b].

O *Repository pattern* serve para criar uma camada de abstração entre a camada de dados e a camada de negócio da aplicação. É um padrão de acesso a dados que leva a uma abordagem mais acoplada no seu acesso. O acesso aos dados é criado num conjunto de classes denominadas repositórios (ver Figura 4.2) [Mic15c].

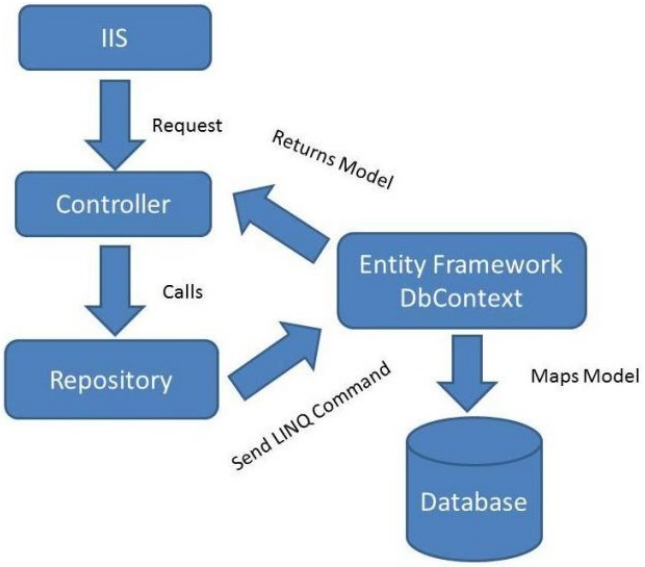


Figura 4.2: *Repository pattern* em MVC [cs13]

4.4 Ferramentas

Todo o código de testes será feito no *Visual Studio*[1], juntamente com o projeto que está a ser desenvolvido, uma vez que assim se torna possível a interação entre as classes e métodos do projeto e os testes.

Optou-se pela utilização de ferramentas e *frameworks open source* por forma a evitar custos, no entanto, na maioria destas ferramentas as opções que existiam eram apenas *open source*. Nos casos em que isto não acontecia não justificava o custo de uma ferramenta paga.

Nos testes que são realizados no *Visual Studio* terá que haver recurso a algumas *frameworks* de testes. Para os exemplos e para as especificações executáveis vai ser utilizado o *SpecFlow*[2]. Esta *framework* é a única que permite integração com ambientes .NET e faz a criação automática das classes e dos métodos para as especificações executáveis, com base em cada passo dos exemplos. Dentro destes métodos são realizados os testes à interface do utilizador através da *framework* Selenium WebDriver. Esta

[1] `https://www.visualstudio.com/`
[2] `http://www.specflow.org/`

ferramenta de automatização do *browser* permite através de chamadas nos métodos de teste uma interação real com o *browser* daquilo que são os passos do utilizador. É possível ainda visualizar cada ação quando os testes são executados. São suportados por esta ferramenta um vasto número de *browsers* incluindo o *Chrome, Firefox, Internet Explorer e Safari*.

Os testes unitários são feitos com recurso à *framework NUnit*[3] que também é destinada às linguagens Microsoft e tem integração no *Visual Studio*.

Para a integração contínua vai ser utilizada a ferramenta *open source Jenkins CI*[4] que é um servidor de integração continua que permite o agendamento de tarefas, incluindo os testes. Esta ferramenta tem disponíveis 985 *plugins* que auxiliam as tarefas.

Nos testes de desempenho e carga, vai ser utilizada a ferramenta *Apache JMeter*[5] destinada a aplicações *web*, desenvolvida pela Apache Software Foundation. A escolha recaiu sobre esta ferramenta visto que todas as outras que foram analisadas implicavam custos elevados, nomeadamente a versão *Ultimate* do *Visual Studio* que disponibiliza ferramentas de testes de desempenho.

[3] http://www.nunit.org/
[4] http://jenkins-ci.org/
[5] http://jmeter.apache.org/

Capítulo 5

Implementação

Este capítulo vai incidir sobre todo o trabalho prático desta dissertação. Inicialmente vai ser dado a conhecer o produto projetado pela *F3M Information Systems, SA* sob o qual vai ser implementado o processo de testes. Depois, vai ser explicada de forma pormenorizada toda a implementação do processo *Behavior-Driven Development*.

5.1 PNG (Produção - Nova Geração)

O projeto sob o qual vai ser implementado o processo de testes é um dos projetos mais recentes da empresa. O objetivo deste projeto é o desenvolvimento de uma solução de gestão que permita a evolução das atuais aplicações da *F3M Information Systems, SA* para o sector da produção têxtil.

Este projeto surge para colmatar o atraso funcional e tecnológico das soluções atuais orientadas para o setor da produção têxtil, se comparadas com outras soluções do setor. Espera-se uma inversão da situação, através desta solução, que passará por novo *software* para dar resposta à evolução necessária nas aplicações deste setor de forma a que se tornem mais dinâmicas, modernas e amigavéis.

Pretende-se com este produto dar resposta aos requisitos funcionais e

necessidades dos clientes atuais e futuros clientes ao nível da gestão de operações e informação estratégica no setor têxtil nacional. Como um sistema de gestão global, procura também trazer uma evolução funcional e tecnológica das aplicações da empresa, com uma possível ligação a outras soluções.

Nesta aplicação, cada secção é orientada para um tema que poderá ser a estrutura global de menus, a definição de uma entidade de dados (artigos, ficha técnica, etc.) ou uma funcionalidade transversal ao sistema (por exemplo processo de compras, contas correntes, etc.).

Os artigos contêm a definição de todas as características gerais inerentes aos movimentos nos diferentes módulos. As características específicas de cada módulo estão organizadas de forma a só aparecerem em função dos licenciamentos e permissões.

5.1.1 Requisitos PNG

Quando começou o trabalho desta dissertação, o projeto sobre o qual é implementado o processo do BDD já tinha sido iniciado. No entanto, os requisitos já estabelecidos foram aproveitados como base para a parte inicial do BDD, onde se definem as *user stories* num formato próprio. Tudo o resto será considerado como se nenhuma funcionalidade estivesse implementada. Mesmo sendo a abordagem do BDD diferente, este também passa pela fase de requisitos tradicional. O *tester* está presente nas reuniões iniciais para a definição dos requisitos, e posteriormente poderá usar estes requisitos, baseados no objetivo de negócio, para escrever as *features* que serão alvo de testes.

Através de conversas com toda a equipa, procurou-se entender de forma mais concreta aquilo que se pretende da aplicação.

De entre os requisitos, foram escolhidos para abordar e implementar com o BDD aqueles que estavam relacionadas com os artigos, uma vez que é um dos conceitos mais importantes neste projeto. Procurou-se então conhecer as características pretendidas desses artigos e quais as

funcionalidades esperadas para estes.

Na página dos artigos pretende-se que estejam disponíveis as seguintes ações:

- Adicionar novo artigo
- Alterar artigo
- Remover artigo
- Exportar o artigo para o formato do Excel
- Imprimir
- Limpar filtros e ordens
- Refrescar a tabela
- Pesquisa e filtragem de artigos por código, descrição e ativação

Cada artigo tem associadas a si informações relativas à sua identificação, definição, composição, stocks, preços, entre outras.

5.2 Implementação do Behavior-Driven Development

Antes de se implementar uma solução de *software* e de serem estabelecidas as funcionalidades a implementar, devem ser entendidos os problemas que se estão a tentar resolver, quem vai utilizar o sistema, o que espera obter dele, e de que forma o sistema vai ajudar os utilizadores ou fornecer valor para as partes envolvidas.

O BDD dá um grande ênfase ao desenvolvimento de *software* que realmente importa e torna os requisitos do cliente em algo que reflete com precisão os valores fundamentais do *software* que o cliente pretende.

Toda a implementação do BDD vai ser detalhada de seguida.

5.2.1 Definição das *features* a testar

Nesta sub-secção vai ser explicado como através dos objetivos de negócio se descrevem as *features* (ver Figura 5.1).

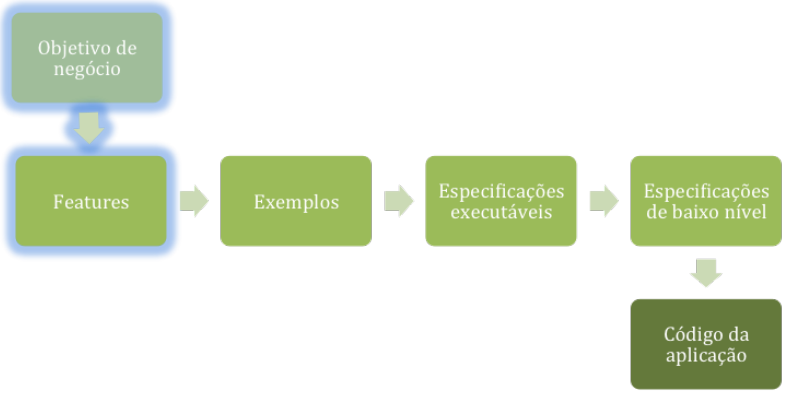


Figura 5.1: Definição do objetivo de negócio e das features

Uma *feature* é parte de uma funcionalidade que terá valor para os utilizadores, está estritamente relacionada com o que os utilizadores pedem e ajuda quer os utilizadores, quer outras partes interessadas a alcançar o objetivo do negócio. As *features* são expressas numa linguagem fácil de entender para elementos responsáveis pelo negócio e pela gestão.

Considere-se o caso dos artigos na aplicação PNG. Nesta nova aplicação, tal como nas anteriores - que se pretende substituir - será possível fazer a gestão de artigos, da área têxtil, por exemplo.

Para escrever as *features* vai ser utilizada a ferramenta *SpecFlow*, como foi referido anteriormente. Esta *feature* pode ser descrita utilizando o formato *"in order to...as a...I want"*, aqui vai ser utilizada a língua Portuguesa. No caso dos artigos referido anteriormente, fica da seguinte forma:

```
Funcionalidade: Gerir artigos
    Para gerir os artigos da fábrica
    Enquanto encarregado pela gestão de aprovisionamento
    Quero ter disponíveis ações para gestão de artigos
```

Através deste formato é possível começar por ter uma breve descrição ou um título da funcionalidade na primeira linha. A segunda linha expressa o objetivo da funcionalidade e permite reavaliar esse objetivo. Na terceira linha identificam-se quais os utilizadores que vão ser afetados por essa funcionalidade. Na última linha descreve-se a funcionalidade e o que realmente significa.

As *features* são *user stories* de alto nível, ou seja, para ir mais ao detalhe estas devem ser "partidas" em *stories* mais pequenas. Enquanto uma *feature* fornece suporte para atingir os objetivos de negócio, uma *user story* permite ir ao detalhe daquilo que se pretende para uma determinada *feature* e servem também como forma de planeamento. No entanto, estas *stories* apenas podem ser criadas quando se estiver perto da implementação dessa *feature*.

Os exemplos para ilustrar as *features* e as *stories* representam bem a forma como o *software* funciona. Estes exemplos deverão ser automatizados na forma de critérios de aceitação automatizados. De seguida vai ser abordado o formato indicado para estruturar esses exemplos e como torná-los em especificações executáveis.

5.2.2 Dos Exemplos às Especificações executáveis

Na sub-secção anterior foi possível ver como podem ser descritas as *features*. Nesta sub-secção vão ser expressos os exemplos referentes à *feature* de gestão de artigos(ver Figura 5.2).

Figura 5.2: Dos exemplos às especificações executáveis

As especificações executáveis são feitas numa linguagem entendida por programadores, gestores de negócio, *testers* ou outras partes interessadas e que facilita também a sua automação. No entanto, quando as conversas começam a abordar demasiados detalhes, que podem ser fulcrais para o funcionamento da aplicação, estas podem ser difíceis de entender. Estas especificações produzem resultados dos testes que reportam o sucesso ou a falha das *features* definidas pelas partes interessadas.

A automatização das especificações pode ser feita recorrendo a diversas ferramentas, neste caso será utilizado o *SpecFlow*. Aqui entra a linguagem *Gherkin* já referida anteriormente, onde se utiliza a estrutura *"Given...When...Then"*.

Na gestão de artigos pretende-se que existam várias funcionalidades, como é sabido da secção anterior, e para cada uma dessas funcionalidades vai ser criado um cenário.

No caso de pesquisar por um artigo o utilizador quando se encontra na página dos artigos, deverá inserir no campo de pesquisa o código do artigo que pretende procurar, e espera-se que esse artigo seja encontrado e que apareça na tabela dos resultados. Este exemplo foi descrito da seguinte forma:

```
Cenario: Pesquisar artigo
    Dado Esta na pagina dos artigos
```

```
    Quando insere <artigo> no campo de pesquisa
    E carrega enter
    Entao o <artigo> deve aparecer nos resultados
```

Quando se trata de um cenário onde é necessário incluir vários exemplos, podem-se utilizar tabelas, sem ter de estar sempre a repetir o mesmo cenário apenas para efetuar ligeiras alterações. Veja-se o caso de filtrar artigos onde a filtragem pode ser feita por código ou por descrição. Para além disso a palavra a filtrar raramente será a mesma, daí que neste caso faça sentido existir uma tabela onde consoante os testes que se pretenda fazer se pode alterar quer o conteúdo quer a coluna a filtrar.

```
Esquema do Cenario: Filtrar artigos
    Dado Esta na pagina dos artigos
    Quando filtra na coluna '<coluna>' a palavra '<palavra>'
    Entao deve aparecer na tabela dos resultados '<palavra>'
    Exemplos:
    | coluna    | palavra         |
    | Codigo    | Artigo1         |
    | Descricao | Descricao teste |
```

Para estes cenários o utilizador terá de fazer um *login* prévio, tal como para qualquer outra funcionalidade da aplicação. Assim, terá de existir um `Contexto`, em inglês `Background`, comum a todos os cenários da *feature* em causa, que vai ser executado sempre em primeiro lugar.

```
Contexto:
Dado O utilizador fez login
```

O aspeto final do ficheiro das *features* é apresentado na figura 5.3.

```
#language: pt-PT
@web
Funcionalidade: Gerir artigos
    Para gerir os artigos da fábrica
    Enquanto encarregado pela gestão de aprovisionamento
    Quer ter disponiveis ações para gestão de artigos

Contexto:
    Dado O utilizador fez login

Cenario: Artigos_Pesquisar artigo
    Dado Esta na pagina dos artigos
    Quando insere 'artigo1' no campo de pesquisa
    E carrega enter
    Entao 'artigo1' deve aparecer nos resultados

Esquema do Cenario: Artigos_Filtrar artigos
    Dado Esta na pagina dos artigos
    Quando filtra na coluna '<coluna>' a palavra '<palavra>'
    Entao '<palavra>' deve aparecer nos resultados
    Exemplos:
    | coluna    | palavra |
    | codigo    | art1    |
    | descricao | malha   |
```

Figura 5.3: Ficheiro *features*

Os restantes exemplos referentes aos artigos encontram-se no Anexo A.1.

De seguida vão ser automatizados os cenários no *Specflow*.

5.2.3 Automatização dos cenários

Nem todos os cenários necessitam de ser automatizados, porque são extremamente complexos para automatizar, a relação custo/benefício não é vantajosa ou porque esse cenário não é tão relevante para o negócio.

Quando um cenário é automatizado traz vários benefícios, a começar pelos testes de regressão uma vez que não terá de haver a preocupação de estar sempre a escrever os mesmos testes cada vez que se termina uma iteração, o que permite que haja um foco maior noutro tipo de testes. Outra das vantagens está relacionada com as novas versões da aplicação que podem ser lançadas mais rapidamente graças à escassez de testes manuais.

Nesta sub-secção vai ser escrito o código que possibilita a automatização dos cenários (ver Figura 5.4).

Figura 5.4: Especificações executáveis - automatização dos cenários

Os cenários referidos anteriormente serão agora automatizados, recorrendo à ferramenta *SpecFlow* onde cada passo do cenário é implementado na linguagem desejada - neste caso em C#.

O *SpecFlow* faz o mapeamento dos passos que constam no cenário e cria automaticamente os respetivos métodos. Para o caso de pesquisa de um artigo referido anteriormente são criados os métodos apresentados na listagem 5.1.

```
[Given(@"O utilizador fez login")]
public void DadoOUtilizadorFezLogin()
{
    ScenarioContext.Current.Pending();
}
[Given(@"Esta na pagina dos artigos")]
public void DadoEstaNaPaginaDosArtigos()
{
    ScenarioContext.Current.Pending();
}
[When(@"insere (.*) no campo de pesquisa")]
public void QuandoInsereNoCampoDePesquisa(string p0)
{
    ScenarioContext.Current.Pending();
```

```
}
[When(@"carrega enter")]
public void QuandoCarregaEnter()
{
    ScenarioContext.Current.Pending();
}
[Then(@"o (.*) deve aparecer nos resultados")]
public void EntaoODeveAparecerNosResultados(string p0)
{
    ScenarioContext.Current.Pending();
}
```

Listagem 5.1: Métodos gerados a partir do cenário

Cada método diz à ferramenta qual o código que deve executar para cada passo do cenário e são extraídas as variáveis que são passadas como parâmetro de cada método, e depois vão ser utilizadas no código de teste.

Na implementação de cada passo dos cenários optou-se por testar a interface uma vez que o BDD é um processo *outside-in*, o que possibilita começar pelo comportamento esperado, e também porque este tipo de testes são uma boa forma de ilustrar e verificar como todos os componentes do sistema funcionam juntos.

Os testes à interface permitem uma boa representação da maneira como o utilizador interage com a aplicação, reproduzem o comportamento do utilizador final, servem como demonstração para os *stakeholders* e reduzem a necessidade de testes manuais à interface. Mais à frente estes testes vão ser complementados com testes unitários - num nível mais baixo - que permitem testar o comportamento da estrutura.

Para a implementação destes cenários vai-se recorrer à ferramenta de automatização *Selenium WebDriver*.

Implementação dos cenários

Pegando no caso da pesquisa de um artigo, referido no exemplo anterior, os métodos serão implementados partindo do princípio que a página terá

o mínimo de detalhes possíveis sobre a implementação. Qualquer detalhe que possa vir a estar presente na página será posteriormente alterado nos métodos de teste.

Antes de realizar qualquer ação dentro da plataforma é necessário fazer *login*. Para isso são preenchidos os campos de utilizador e *password* e depois a ação de carregar no botão de *login* (ver listagem 5.2).

```
[Given(@"O utilizador fez login")]
public void DadoOUtilizadorFezLogin()
{
    selenium.NavigateTo("http://*****.*****.net:8081/
        Seguranca/Autentica/Login");

    selenium.FindElement(By.Id("UserName")).SendKeys("f3m
        ");
    selenium.FindElement(By.Id("Password")).SendKeys("
        pass");
    selenium.FindElement(By.Id("btnAutenticacao")).Click
        ();
}
```

Listagem 5.2: Automatização do *login*

O primeiro passo do cenário é uma pré-condição para a pesquisa de artigos, neste caso é necessário estar na página dos artigos e como tal é utilizado o método *NavigateTo()* (ver listagem 5.3).

```
[Given(@"Esta na pagina dos artigos")]
public void DadoEstaNaPaginaDosArtigos()
{
        selenium.NavigateTo("http://*****.*****.net:8081/
            Producao/Artigos/Artigos");
}
```

Listagem 5.3: Automatização da entrada nos artigos

Depois de se estar na página dos artigos e como se pretende fazer uma pesquisa de um artigo deve-se introduzir um valor no campo de pesquisa.

Para preencher o campo de pesquisa é necessário primeiro encontra-lo, e para tal o *Selenium* utiliza ao método *FindElement()* que vai procurar o campo através de um atributo, neste caso é o *Id*. Juntamente com o método *FindElement()* é utilizado o método *SendKeys()* que simula a escrita e vai preencher o campo de pesquisa com o valor que foi atribuído a *p0* no cenário definido (ver listagem 5.4)

```
[When(@"insere (.*) no campo de pesquisa")]
public void QuandoInsereNoCampoDePesquisa(string p0)
{
    selenium.FindElement(By.Id("Pesquisa")).SendKeys(p0);
}
```

Listagem 5.4: Automatização de preenchimento da pesquisa

Após o preenchimento do campo de pesquisa com o valor pretendido terá de se premir a tecla *Enter* para a pesquisa surtir efeito. O código é idêntico ao do exemplo anterior exceto no simulador do teclado, que neste caso vai premir o *Enter* (ver listagem 5.5).

```
[When(@"carrega enter")]
public void QuandoCarregaEnter()
{
    selenium.FindElement(By.Id("Pesquisa")).SendKeys(
        OpenQA.Selenium.Keys.Enter);
}
```

Listagem 5.5: Automatização para confirmar a pesquisa

Por fim, para confirmar que a pesquisa funciona conforme o esperado, todos os resultados obtidos são inseridos numa lista e cada entrada dessa lista será comparada com o valor que foi alvo de pesquisa. Caso o valor seja encontrado, significa que o teste foi bem sucedido; caso a lista esteja vazia, ou não contenha o valor procurado, significa que o teste falhou (ver listagem 5.6).

```
[Then(@"o (.*) deve aparecer nos resultados")]
```

```
public void EntaoODeveAparecerNosResultados(string p0)
{
        Thread.Sleep(tempo);
    IList<IWebElement> tArt = new List<IWebElement>();
    IWebElement tableRowCollection = selenium.FindElement
        (By.XPath("//*[@id='F3MGrelhaFormArtigos']/div
        [2]/table/tbody"));

    tArt = tableRowCollection.FindElements(By.XPath("//*[
        @id='F3MGrelhaFormArtigos']/div[2]/table/tbody/tr
        /td"));
    bool isEmpty = !tArt.Any();
    if (isEmpty)
    {
        Assert.Fail("Nao existe");
    }
    else {
        foreach (IWebElement row in tArt)
        {
            if (row.Text.Equals(p0))
            {
                Assert.AreEqual(p0, row.Text);
            }
        }
    }
}
```

Listagem 5.6: Automatização de verificação do resultado

As restantes especificações executáveis referentes aos artigos encontram-se no Anexo A.2.

5.2.4 Especificações de baixo nível - Testes unitários

O BDD ainda vai mais além dos requisitos e dos testes de aceitação automatizados. Este processo, através de práticas comuns ao TDD - especificações de baixo nível ou testes unitários - ocorre também de uma forma mais direta no desenvolvimento do código da aplicação, servindo como

guia durante a implementação (ver Figura 5.5).

Figura 5.5: Especificações de baixo nível

O BDD praticado num nível mais baixo, é a continuação dos princípios já aplicados aos requisitos num nível mais alto. Enquanto que os requisitos de alto nível retratam o comportamento do sistema de uma forma mais generalizada e no âmbito do negócio, os requisitos de baixo nível tratam do comportamento das classes ou partes do código, sob um ponto de vista mais técnico e destinado ao programadores.

Um dos princípios fundamentais do BDD é o desenvolvimento de fora para dentro, ou seja, o processo inicia-se com o que se prevê que seja o resultado esperado que servirá para determinar o código a desenvolver, e gradualmente vai-se baixando o nível de abstração.

Quando se começam a fazer os testes unitários é quando surge o primeiro contacto com a aplicação, portanto, já deve haver uma noção de qual será a sua arquitetura e o seu desenho geral. No PNG, como já foi referido, utiliza-se a arquitetura MVC com *Repository patterns* que trabalham com a *Entity Framework*. Será criado um repositório para cada entidade e um repositório genérico onde vão estar presentes os métodos genéricos. Nos repositórios das entidades constam os métodos específicos dessa entidade e os métodos herdados pelo repositório geral. O repositório genérico vai ter ainda uma interface.

O código dos testes unitários deve ser aquele que se deseja ter du-

rante a implementação, imaginando as classes e os métodos necessários. Olhando para o caso do PNG, e pegando na funcionalidade de adicionar artigos vai ser testado o `RepositorioArtigos` visto que é através dela que pode ser testada a lógica de negócio.

Inicialmente foi instanciado um novo artigo, artigo esse que depois foi adicionado à base de dados através do método `Adicionar()` do `RepositorioArtigos`, instanciado como `repo`. No final verificou-se o resultado esperado, ou seja, que o artigo com o código "artigo01" foi adicionado com sucesso. É de notar que foi utilizada a classe `TransactionScope` para garantir que não houve qualquer inserção na base de dados, e desta forma que não existem dependências externas (ver listagem 5.7).

```
[TestFixture]
class Artigos_unit
{
    IRepositorioGenerico<tbArtigos, Artigos> repo = new
        RepositorioArtigos();

    tbArtigos art = new tbArtigos()
    {
        Codigo = "artigo_01",
        Descricao = "ardd323tt",
        IDTipoArtigo = 1,
        IDGrupo = 1,
        IDFamilia = 1,
        IDMarca = 2,
        IDSubFamilia = 1,
        IDUnidade = 2,
        IDUnidadeCompra = 3,
        IDUnidadeVenda = 3,
        UtilizadorCriacao = "F3M",
        DataCriacao = DateTime.Now,
    };

   [Test]
    public void should_add()
    {
```

```
        using (TransactionScope scope = new
            TransactionScope())
        {
            repo.Adiciona(art);
            repo.Grava();

            Assert.AreEqual(art.Codigo, repo.
                ObtemPorObjID(art.ID).Codigo);
        }
    }
}
```

Listagem 5.7: Especificação de baixo nível

Assim que o código da aplicação for implementado os testes unitários podem ser revistos, principalmente nomes de classes, métodos ou variáveis.

Em seguida estes testes vão ser executados através da ferramenta de testes unitários NUnit.

As restantes especificações de baixo nivel referentes aos artigos encontram-se no Anexo A.3.

5.3 Organização e Execução

Todo o processo até este ponto desenvolveu-se no *Visual Studio*, o que possibilitou que se recorresse a várias ferramentas no apoio aos testes.

O BDD requer uma boa organização dos ficheiros dentro do projeto de testes. Cada ficheiro `.feature` diz respeito apenas a uma *feature*, que se divide em vários cenários. Cada cenário desta *feature* vai dar origem a um ficheiro `.cs` onde são automatizados os cenários referentes a essa *feature*. Os testes unitários, num formato `.cs` também estão separados da mesma forma apesar de aqui não existir qualquer relação entre os ficheiros das *features*. Quanto à organização das pastas no projeto, existem três pastas onde estão todos os testes, sendo que uma pasta é

para as *features*, outra para os testes à interface web e ainda uma para os testes unitários. Na Figura 5.6 é possível observar a estrutura de pastas utilizada neste projeto:

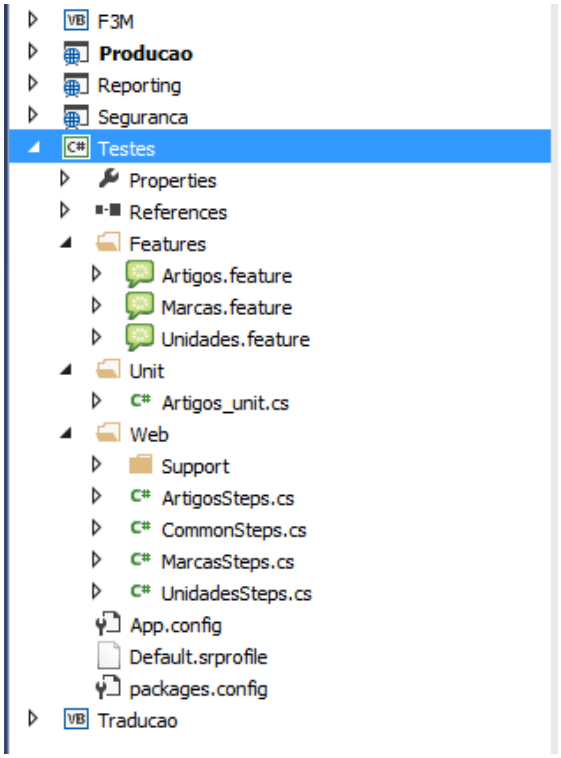


Figura 5.6: Estrutura de pastas no projeto de testes

A execução dos testes referidos anteriormente pode ser feita através do *Visual Studio*. Assim que os testes forem implementados e for feito *build* do projeto, a janela *Test Explorer* apresenta todos os testes disponíveis para execução. A execução dos testes acontece por via do *MSTest* e logo que o teste for executado será apresentado o resultado com três estados possíveis - sucedido, falhado ou inconclusivo - e o respetivo tempo de execução. Na Figura 5.7 é apresentada essa mesma janela com alguns exemplos de testes efetuados.

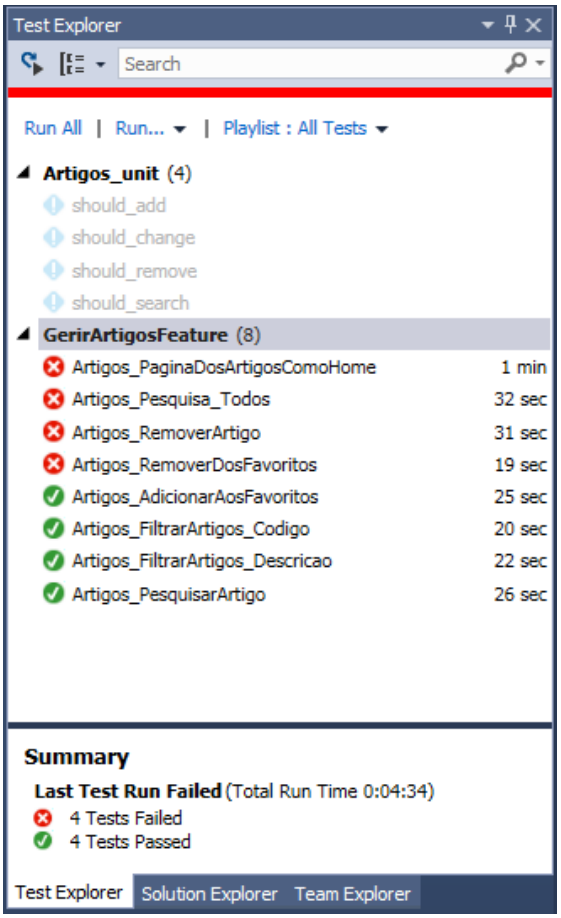


Figura 5.7: Janela de testes

No final da execução dos testes são gerados ainda relatórios em formato HyperText Markup Language (HTML) com informações mais detalhadas sobre o teste. Estes relatórios vão ser abordados na Secção 5.4.

5.4 Documentação: Relatórios

Os relatórios do BDD descrevem o que se espera que a aplicação faça e reportam se o está a fazer corretamente ou não. Estes relatórios, gerados pelos testes automatizados combinam as especificações, os critérios de aceitação e os resultados dos testes. Completam o ciclo do BDD, possibilitando que os *stakeholders*, os analistas ou os *testers*, possam visualizar as conversas e os exemplos discutidos na fase inicial do processo.

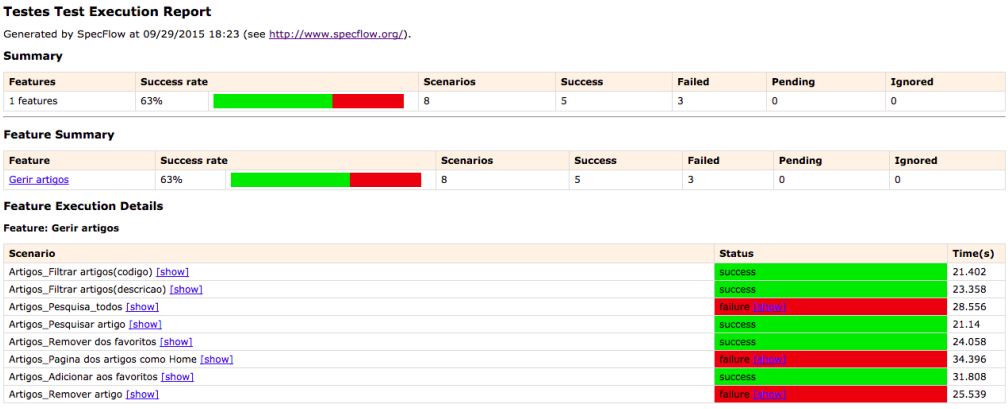
A informação presente nos relatórios é também importante no trabalhos dos *testers*, por complementar as atividades de testes desenvolvidas de forma a perceber melhor quais as partes que necessitam de uma maior atenção durante os testes exploratórios.

Após o processo natural do BDD, sempre que os testes são executados, podem ser gerados estes relatórios recorrendo às ferramentas de testes utilizadas e já referidas anteriormente. Neste caso utilizou-se a ferramenta *Specflow* que gera o relatório em formato HTML e que contém um sumário sobre os testes executados, o seu resultado e ainda cada cenário detalhado de forma individual.

Para obter estes relatórios é necessário executar o seguinte comando que, através do ficheiro de resultados criado pelo *Visual Studio* vai gerar o ficheiro HTML.

```
specflow.exe mstestexecutionreport Testes.csproj /
    testResult:result.trx /out:MyResult.html
```

O ficheiro HTML gerado tem a apresentação visível na figura 5.8.



Testes Test Execution Report

Generated by SpecFlow at 09/29/2015 18:23 (see http://www.specflow.org/).

Summary

Features	Success rate	Scenarios	Success	Failed	Pending	Ignored
1 features	63%	8	5	3	0	0

Feature Summary

Feature	Success rate	Scenarios	Success	Failed	Pending	Ignored
Gerir artigos	63%	8	5	3	0	0

Feature Execution Details

Feature: Gerir artigos

Scenario	Status	Time(s)
Artigos_Filtrar artigos(codigo) [show]	success	21.402
Artigos_Filtrar artigos(descricao) [show]	success	23.358
Artigos_Pesquisa_todos [show]	failure [throw]	28.556
Artigos_Pesquisar artigo [show]	success	21.14
Artigos_Remover dos favoritos [show]	success	24.058
Artigos_Pagina dos artigos como Home [show]	failure [throw]	34.396
Artigos_Adicionar aos favoritos [show]	success	31.808
Artigos_Remover artigo [show]	failure [throw]	25.539

Figura 5.8: Estrutura do relatório

Como é possível observar, o documento tem uma apresentação aprazível e mostra os resultados dos testes de forma sintetizada. Caso se pretenda saber mais pormenores sobre um cenário em concreto, basta clicar no respetivo *link* que irá redirecionar para os passos do cenário que, em

caso de insucesso, apresenta e localiza o erro, como mostra a Figura 5.9:

Figura 5.9: Relatório de cenário com erro

5.5 Testes de *Performance*

Os testes de *Performance* servem para testar alguns dos requisitos não funcionais, como a capacidade de resposta, estabilidade e escalabilidade de uma aplicação sob condições de alta carga.

O desempenho é uma das principais preocupações do utilizador final de uma aplicação. Como tal, neste tipo de testes o comportamento esperado continua a ser o foco principal, assim como no processo do BDD. Neste projeto, tratando-se de uma aplicação onde se espera que haja muita concorrência de acessos, é essencial que os tempos de resposta sejam considerados aceitáveis para os utilizadores.

Para testar o desempenho da aplicação em questão foi utilizada a ferramenta *JMeter*. Esta ferramenta grava um passo que se pretenda analisar e permite correr posteriormente essa ação com diferentes configurações.

Foi realizado um teste com 50 utilizadores a fazerem *login* na aplicação ao mesmo tempo de forma a perceber quais seriam os pontos mais críticos. A Figura 5.10 mostra todos os pedidos que são feitos durante o *login*.

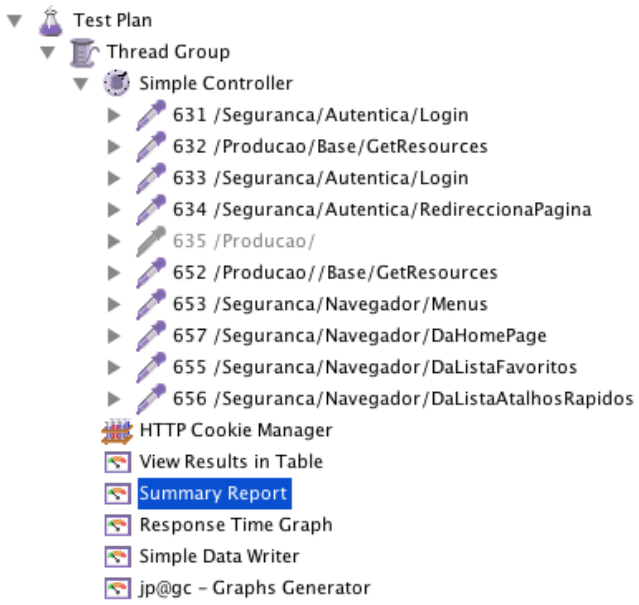


Figura 5.10: HTTP Requests durante o login

No final da execução foram gerados sumários e gráficos com os dados obtidos por forma a facilitar a análise do desempenho, apresentados nas Figuras 5.11 e 5.12.

Label	# Samples	Average	Min	Max	90% Line	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
631 /Seguranca/Autentica/Login	50	198	174	481	217	47.83	0.00%	5.0	35.95	7414.0
632 /Producao/Base/GetResources	50	79	59	422	85	54.10	0.00%	5.1	21.17	4286.0
633 /Seguranca/Autentica/Login	50	1412	408	2275	1896	462.93	0.00%	4.5	125.61	28608.0
634 /Seguranca/Autentica/RedireccionaPagina	50	150	125	290	203	36.72	0.00%	4.8	128.41	27344.0
652 /Producao//Base/GetResources	50	70	59	154	78	17.48	0.00%	4.8	20.26	4286.0
653 /Seguranca/Navegador/Menus	50	267	157	563	329	75.28	0.00%	4.8	43.76	9384.0
657 /Seguranca/Navegador/DaHomePage	50	97	61	236	185	43.37	0.00%	4.8	1.25	265.0
655 /Seguranca/Navegador/DaListaFavoritos	50	111	63	544	175	74.49	0.00%	4.8	9.50	2022.0
656 /Seguranca/Navegador/DaListaAtalhosRapidos	50	98	61	245	134	38.43	0.00%	5.0	6.69	1374.0
TOTAL	450	276	59	2275	563	437.37	0.00%	37.0	341.02	9442.6

Figura 5.11: Sumário da execução do teste de performance

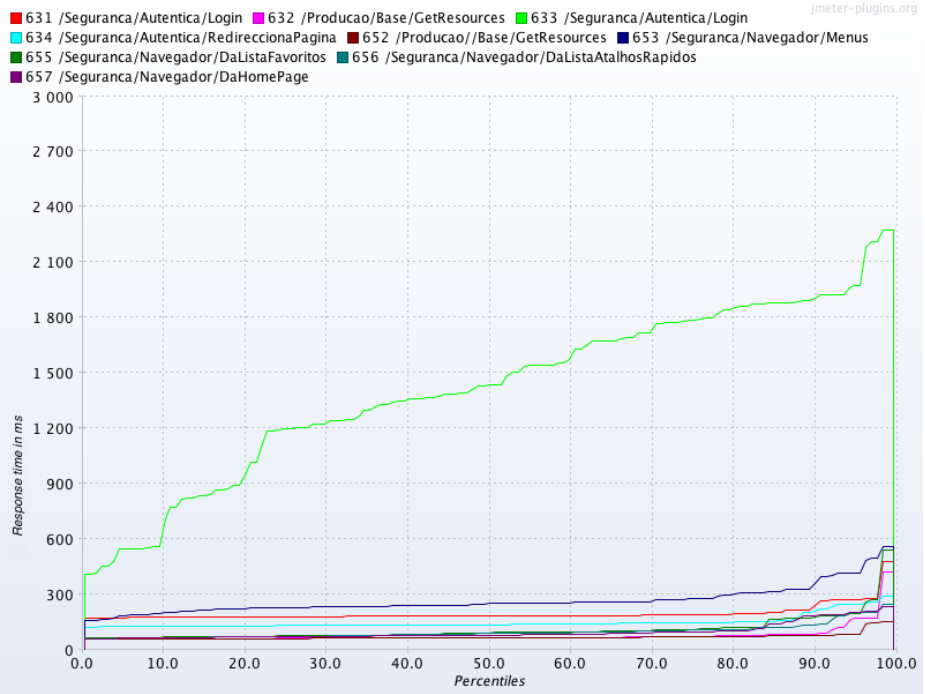


Figura 5.12: Gráfico Tempos de resposta/Percentagem de execução

A interpretação destes resultados é feita maioritariamente com base nos tempos de resposta. Para tal, recorreu-se ao trabalho de Jakob Nielsen [Nie93] onde são identificados os seguintes limites, relacionados com a natureza do ser humano:

- 0.1 segundo - o sistema reage instantaneamente;
- 1 segundo - limite de tempo para o pensamento se manter ininterrupto;
- 10 segundos - limite de tempo para o manter a atenção do utilizador.

Através destes limites é possível ter uma orientação mais precisa durante os testes de desempenho.

Assim, dado que os resultados são apresentados em milissegundos, pode-se verificar que os tempos de resposta obtidos se encontram dentro dos limites aceitáveis de espera.

5.6 Testes exploratórios

No decorrer de um processo de testes o trabalho é feito para que tudo funcione conforme o esperado, no entanto, mesmo que se tenha trabalhado da forma mais afincada possível irá haver sempre um exemplo mal enquadrado, uma especificação mal definida ou um teste bem sucedido que não era suposto ser. A perceção que se tem de um produto quando ainda está em fase de desenvolvimento é diferente de quando se sente e interage com esse produto no seu formato final. É algo que é impossível de automatizar, diz respeito apenas ao intelecto humano e ao instinto.

Para fazer os testes exploratórios são aplicadas técnicas e heurísticas de forma disciplinada, aliadas a um vasto conhecimento da vida real, pensamento critico e uma boa interpretação de resultados.

Este tipo de testes não significa que o *software* é testado exaustivamente, mas sim o necessário para confirmar se o que foi pensado está realmente concretizado ou se necessita de melhorias ou novas funcionalidades.

Quanto à aplicação PNG, os testes exploratórios procuraram sobretudo clarificar a ideia geral que foi alimentada no decorrer do desenvolvimento. É certo que, agora que parte da aplicação se encontra operacional, surjam melhorias a incorporar nos testes automatizados, assim como novos cenários de testes. No entanto, visto que os testes exploratórios não implicam o uso de nenhuma ferramenta ou método e que recorre muito à experiência e ao poder de decisão, vão existir sempre certos aspetos num produto que não dizem respeito apenas a quem está encarregue pelos testes. Questões relacionadas com design gráfico ou de ordenação, entre outras, são exemplos disso.

5.7 Integração contínua

O *feedback* é um dos princípios fundamentais nas práticas de desenvolvimento ágil. Sendo este rápido, pode ser uma forma de evitar o desper-

dício de recursos, e tornando as entregas mais eficientes.

A integração continua surge precisamente para proporcionar um *feedback* de forma rápida e atempada, evitando riscos associados a possíveis atrasos, ou verificar que a estratégia adotada é a correta. É uma prática que permite que o *build* e os testes de um projeto sejam feitos de forma automática e é essencial no alerta para potenciais erros de integração ou regressões. No entanto, exige um conjunto abrangente de testes automatizados.

Para que a integração contínua possa monitorizar um projeto, é necessário que haja um servidor. Este servidor é configurado para que sempre que for registado um *commit* de qualquer alteração sejam despoletados o *build* e os testes no projeto, sem que haja qualquer preocupação em ter de o fazer manualmente.

A aplicação de integração contínua utilizada foi o *Jenkins*. Esta aplicação através do seu extenso número de *plugins* permite lidar com praticamente todos os tipos de projetos, independentemente do ambiente em que foi desenvolvido. Foram utilizados os seguintes *plugins*:

- *MSBuild* para fazer *build* do projeto;
- *MSTest* para executar os testes;
- *Selenium plugin* para os testes à interface;
- *Performance plugin* para executar os testes de *performance*;
- HTML *Report* para gerar relatórios no formato HTML.

Após o agendamento de um *build* e a posterior execução dos testes são apresentados num *dashboard* os resultados dos testes das *features*, sob a forma de relatório (num formato igual ao já aqui mostrado em 5.4), e dos testes de performance com tabelas e gráficos idênticos aos apresentados em 5.5.

O *Jenkins* foi também configurado para enviar email com os resultados no final de cada agendamento de tarefas.

Capítulo 6

Conclusões e trabalho futuro

6.1 Conclusões

Neste trabalho sobre testes de *software* foi realizada a análise e a implementação de um processo de testes que a *F3M Information Systems, SA* carecia.

Todo o estudo foi realizado tendo em conta muitas das práticas já utilizadas na empresa, principalmente a metodologia de desenvolvimento de *software*, que tem sempre um papel relevante na escolha de um processo de testes. Outro dos aspetos tidos em conta foi evitar que as novas práticas a adotar implicassem muitas alterações pois tornaria o processo mais difícil de implementar.

O processo escolhido, *Behavior-Driven Development*, é extremamente atual e inovador na forma como encara as necessidades do cliente e todo o processo de desenvolvimento. Trata-se de práticas que se têm afirmado como um ponto de viragem no que concerne ao desenvolvimento e testes de *software* e que terão cada vez mais um papel fundamental para que sejam cumpridas todas as expectativas dos clientes, que tendem a exigir cada vez mais qualidade nos produtos.

O BDD é um processo que se baseia essencialmente em três princípios: descrever o comportamento, descobrir o comportamento que acres-

centa valor ao negócio e utilizar conversas e exemplos para se referir ao que é esperado do sistema. A automatização dos exemplos que representam os requisitos é também de realçar, pois torna-se muito vantajoso por permitir uma brutal redução do tempo consumido em testes que eram feitos manualmente.

Sendo a redução do tempo gasto nos testes e a redução do número de falhas no *software* dois dos principais objetivos de quem pretende implementar um processo de testes, estes requerem que haja um maior investimento inicial. Portanto, num processo rigoroso e estruturado como o BDD, é necessário dedicar mais tempo aos testes na sua fase inicial face aos processos tradicionais, no entanto, no decorrer do desenvolvimento o tempo gasto nos testes é muito reduzido. Quanto à redução das falhas, apenas se poderá verificar num médio/longo prazo, mas é certo que haverá uma grande redução. A mudança de hábitos que o BDD requer durante o desenvolvimento, também pode causar alguma estranheza por parte da equipa nos primeiros tempos, no entanto, é uma reação normal para qualquer prática que seja introduzida pela primeira vez visto que o ser humano é avesso à mudança.

Assim, com todo o trabalho que foi desenvolvido ao longo desta dissertação foi possível criar de raiz um processo baseado no BDD que se adequou às necessidades da empresa. Conclui-se que este trabalho contribuiu e foi importante para a empresa pois o processo vai ser adotado no imediato e vai ser possível colocar em prática todos os conhecimentos adquiridos.

6.2 Trabalho futuro

Como se tratou de algo novo, primeiro foi testado todo o processo nas aplicações da empresa com alguns membros da equipa, mas sem interferir diretamente com o trabalho que é feito atualmente nessas mesmas aplicações. No entanto, foram adquiridos os conhecimentos e a familiarização necessária para que seja dada a continuidade a este trabalho

durante os próximos tempos.

Prevê-se durante esse período consolidar e melhorar alguns aspetos, tais como a adaptação do processo a aplicações mais antigas e que ainda são importantes para a *F3M Information Systems, SA* e a forma como é feita a recolha dos requisitos e a sua ligação com o BDD.

Bibliografia

[Adz09] Gojko Adzic. *Bridging the communication gap: specification by example and agile acceptance testing*. Neuri Limited, 2009.

[Bec01] Jeff Beck, K. and Beedle, M. and van Bennekum, A. and Cockburn, A. and Cunningham, W. and Fowler, M. and Grenning, J. and Highsmith, J. and Hunt, A. and Jeffries, R. and Kern, Jon and Marick, Brian and Martin, Robert C. and Mallor, Steve and Shwaber, Ken and S. Manifesto for Agile Software Development, 2001.

[Bec03] Kent Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[BK02] R Black and G Kubaczkowski. Mission made possible automating unit, component, and integration testing can sometimes seem like an impossible mission. *SOFTWARE TESTING AND QUALITY ENGINEERING*, 4:42–49, 2002.

[CG09] Lisa Crispin and Janet Gregory. *Agile testing: A practical guide for testers and agile teams*. Pearson Education, 2009.

[Coh04] Mike Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[Cop04] Lee Copeland. *A practitioner's guide to software test design*. Artech House, 2004.

[cs13] c sharpcorner. Crud using the repository pattern in mvc, 2013.

[GVVE08] Dorothy Graham, Erik Van Veenendaal, and Isabel Evans. *Foundations of software testing: ISTQB certification*. Cengage Learning EMEA, 2008.

[Hen08] Elisabeth Hendrickson. Driving development with tests: Atdd and tdd. *STARWest 2008*, 2008.

[JS05] David Janzen and Hossein Saiedian. Test-driven development: Concepts, taxonomy, and future direction. *Computer*, (9):43–50, 2005.

[Lak09] Lakeworks. Scrum process, 2009.

[Lin03] Johannes Link. *Unit testing in Java: how tests drive the code*. Morgan Kaufmann, 2003.

[Mic15a] Microsoft. Asp.net and visual studio for web, 2015.

[Mic15b] Microsoft. Asp.net mvc overview, 2015.

[Mic15c] Microsoft. The repository pattern, 2015.

[MSB11] Glenford J Myers, Corey Sandler, and Tom Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[Nie93] Jakob Nielsen. Response times: The 3 important limits. *Usability Engineering*, 1993.

[SB02] K Schwaber and Mike Beedle. Agile software development with scrum. 2002.

[Sch04] Ken Schwaber. *Agile project management with Scrum*. Microsoft Press, 2004.

[Sma14] John Ferguson Smart. Bdd in action. 2014.

[SS11] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum. org, October*, 2011.

[TN86] Hirotaka Takeuchi and Ikujiro Nonaka. The new new product development game. *Harvard business review*, 64(1):137–146, 1986.

Apêndice A

Anexo

A.1 *Features* e exemplos

```
#language: pt-PT
@web

Funcionalidade: Gerir artigos
    Para gerir os artigos da áfbrica
    Enquanto encarregado pela ãgesto de aprovisionamento
    Quer ter disponiveis çõaes para ãgesto de artigos

Contexto:
    Dado O utilizador fez login

Cenario: Artigos_Pesquisar artigo
    Dado Esta na pagina dos artigos
    Quando insere 'artigo1' no campo de pesquisa
    E carrega enter
    Entao 'artigo1' deve aparecer nos resultados

Esquema do Cenario: Artigos_Filtrar artigos
    Dado Esta na pagina dos artigos
    Quando filtra na coluna '<coluna>' a palavra '<palavra>'
    Entao '<palavra>' deve aparecer nos resultados
    Exemplos:
    | coluna    | palavra       |
    | codigo    | Artigo1       |
    | descricao | malha acabada |

Esquema do Cenario: Artigos_Adicionar artigo e sair
    Dado Esta na pagina dos artigos
```

```
    Quando seleciona adicionar novo artigo
    E insire os campos '<codigo>' '<descricao>' '<descAbre>'
    E na identificacao '<tipo>' '<familia>' '<subfam>' '<grupo>' '<marca
        >' '<comp>'
    Entao '<codigo>' deve aparecer nos resultados
    Exemplos:
    |codigo|descricao|descAbre|tipo|familia|subfam|grupo|marca|comp|
    |artigo024|desc_teste|desc|tipo1|óAcessrio|Jersey|2|marca1|algodao|
    |artigo_\1|desc_??!%|<''*|tipo4|familia23|subfam|4|marca1|nylon|

Cenario: Artigos_Remover artigo
    Dado Esta na pagina dos artigos
    Quando seleciona o artigo 'Malha Acabada 2' e remove
    Entao o artigo 'Malha Acabada 2' deve desaparecer da tabela

Cenario: Artigos_Pesquisa_todos
    Dado Esta na pagina dos artigos
    Quando insere '*' no campo de pesquisa
    E carrega enter
    Entao na combo das vistas deve aparecer 'Resultados da Pesquisa'
    E nos resultados devem aparecer todos os artigos

Cenario: Artigos_Limpar filtros e ordens
    Dado Esta na pagina dos artigos
    Quando seleciona ordenar nos artigos o 'tipo' por ordem 'decrescente
        '
    E filtra na coluna 'codigo' a palavra 'artigo'
    E seleciona limpar filtros e ordens
    Entao a pagina dos artigos volta ao estado inicial

Cenario: Artigos_Pesquisa e limpeza no final
    Dado Esta na pagina dos artigos
    Quando insere 'artigo1' no campo de pesquisa
    E carrega enter
    E limpa a pesquisa e é feita nova procura sem nada
    Entao a vista é alterada para o default
```

A.2 Especificações executáveis

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
```

```
using System.Windows.Forms;
using TechTalk.SpecFlow;
using Testes.Web.Support;

namespace Testes.Web
{
    [Binding, Scope(Tag = "web")]
    public class GerirArtigosSteps : SeleniumStepsBase
    {
        int tempo = 2000;
        IJavaScriptExecutor js = selenium as IJavaScriptExecutor;
        var strJS = "";

        [Given(@"O utilizador fez login")]
        public void DadoOUtilizadorFezLogin()
        {
            selenium.NavigateTo("http://*****.*****.net:8081/Seguranca/
                Autentica/Login");
            Thread.Sleep(1000);
            selenium.FindElement(By.Id("UserName")).SendKeys("f3m");
            selenium.FindElement(By.Id("Password")).SendKeys("******");
            selenium.FindElement(By.Id("btnAutenticacao")).Click();
        }

        [Given(@"Esta na pagina dos artigos")]
        public void DadoEstaNaPaginaDosArtigos()
        {
            Thread.Sleep(1000);
            selenium.NavigateTo("http://*****.*****.net:8081/Producao/
                Artigos/Artigos");
        }

        [When(@"insere '(.*)' no campo de pesquisa")]
        public void QuandoInsereNoCampoDePesquisa(string p0)
        {
            selenium.FindElement(By.Id("Pesquisa")).SendKeys(p0);
        }

        [When(@"carrega enter")]
        public void QuandoCarregaEnter()
        {
            selenium.FindElement(By.Id("Pesquisa")).SendKeys(OpenQA.
                Selenium.Keys.Enter);
        }

        [When(@"filtra na coluna '(.*)' a palavra '(.*)'")]
        public void QuandoFiltraNaColunaAPalavra(string p0, string p1)
        {
            if (p0.Equals("codigo")) { selenium.FindElement(By.Id("
                Codigo")).SendKeys(p1); }
```

```
    if (p0.Equals("tipo")) { selenium.FindElement(By.Id("Tipo"))
        .SendKeys(p1); }
    if (p0.Equals("descricao")) { selenium.FindElement(By.Id("
        Descricao")).SendKeys(p0); }

    selenium.FindElement(By.Id(p0)).SendKeys(OpenQA.Selenium.
        Keys.Enter);
}

[When(@"seleciona adicionar novo artigo")]
public void QuandoSelecionaAdicionarNovoArtigo()
{
    Thread.Sleep(tempo);
    selenium.FindElement(By.XPath("//*[@id='
        F3MGrelhaFormArtigosBtAdd']")).Click();
}

[When(@"insire os campos '(.*)' '(.*)' '(.*)'")]
public void QuandoInsireOsCampos(string p0, string p1, string p2
    )
{
    Thread.Sleep(tempo);

    selenium.FindElement(By.Id("Codigo")).SendKeys(p0);
    selenium.FindElement(By.Id("Descricao")).SendKeys(p1);
    selenium.FindElement(By.Id("DescricaoAbreviada")).SendKeys(
        p2);
}

[When(@"na identificacao '(.*)' '(.*)' '(.*)' '(.*)' '(.*)'
    '(.*)'")]
public void QuandoNaIdentificacao(string p0, string p1, string
    p2, int p3, string p4, string p5)
{
    Thread.Sleep(tempo);

    strJS = "$('#IDTipoArtigo').data('kendoComboBox').dataSource
        .read(); setTimeout(function(){ $('#IDTipoArtigo').data
        ('kendoComboBox').text('" + p0 + "') }, 100);";
    strJS += "$('#IDFamilia').data('kendoComboBox').dataSource.
        read(); setTimeout(function(){ $('#IDFamilia').data('
        kendoComboBox').text('" + p1 + "')}, 100); ";
    strJS += "$('#IDSubFamilia').data('kendoComboBox').
        dataSource.read(); setTimeout(function(){ $('#
        IDSubFamilia').data('kendoComboBox').text('" + p2 + "')
        },100);";
    strJS += "$('#IDGrupo').data('kendoComboBox').dataSource.
        read(); setTimeout(function(){$('#IDGrupo').data('
        kendoComboBox').text('" + p3 + "')},100);";
```

```
        strJS += "$('#IDMarca').data('kendoComboBox').dataSource.
            read();setTimeout(function(){ $('#IDMarca').data('
            kendoComboBox').text('" + p4 + "')},100);";
        strJS += "$('#IDComposicao').data('kendoComboBox').
            dataSource.read(); setTimeout(function(){ $('#
            IDComposicao').data('kendoComboBox').text('" + p5 + "')
            },100);";

        js.ExecuteScript(strJS);

        selenium.FindElement(By.Id("F3MGrelhaFormArtigosBtSaveFecha"
            )).Click();
    }

    [When(@"seleciona o artigo '(.*)' e remove")]
    public void QuandoSelecionaOArtigoERemove(string p0)
    {
        Thread.Sleep(tempo);
        strJS = "$('#Pesquisa').val('" + p0 + "')";
        js.ExecuteScript(strJS);
        selenium.FindElement(By.Id("Pesquisa")).SendKeys(OpenQA.
            Selenium.Keys.Enter);

        Thread.Sleep(tempo);
        IList<IWebElement> tArt = new List<IWebElement>();
        IWebElement tableRowCollection = selenium.FindElement(By.
            XPath("//*[@id='F3MGrelhaFormArtigos']/div[2]/table/
            tbody"));

        int pages = selenium.FindElements(By.XPath("//*[@id='
            F3MGrelhaFormArtigos']/div[3]/ul/li")).Count;
        int i = 1;
        while (i <= pages)
        {
            tArt = tableRowCollection.FindElements(By.XPath("//*[@id
                ='F3MGrelhaFormArtigos']/div[2]/table/tbody/tr/td
                [2]"));
            Boolean isEmpty = tArt.Any();
            if (!isEmpty)
            {
                Assert.Fail("ãNo existe");
            }

            foreach (IWebElement row in tArt)
            {
                if (string.Equals(p0, row.Text, StringComparison.
                    CurrentCultureIgnoreCase))
                {
                    row.Click();
                    Thread.Sleep(500);
```

```
                    selenium.FindElement(By.XPath("//*[@id='BtRemove
                        ']")).Click();
                    Thread.Sleep(500);
                    selenium.FindElement(By.ClassName("
                        ZebraDialog_Button_1")).Click();
                    Thread.Sleep(500);
                    break;
                }
                else
                {
                    Assert.Fail("Nao foi encontrado o artigo
                        selecionado");
                }
            }
            i++;
        }
    }

    [When(@"limpa a pesquisa e é feita nova procura sem nada")]
    public void QuandoLimpaAPesquisaEEFeitaNovaProcuraSemNada()
    {
        strJS = "$('#Pesquisa').val('')";
        js.ExecuteScript(strJS);
        selenium.FindElement(By.XPath("//*[@id='Pesquisa']")).
            SendKeys(OpenQA.Selenium.Keys.Enter);
    }

    [When(@"seleciona ordenar nos artigos o '(.*)' por ordem '(.*)'"
        )]
    public void QuandoSelecionaOrdenarNosArtigosOPorOrdem(string p0,
        string p1)
    {
        if (string.Equals(p0, "Codigo", StringComparison.
            CurrentCultureIgnoreCase))
        {
            if (string.Equals(p1, "decrescente", StringComparison.
                CurrentCultureIgnoreCase))
            {
                selenium.FindElement(By.XPath("//*[@id='
                    F3MGrelhaFormArtigos']/div[1]/div/table/thead/
                    tr[1]/th[1]/a")).Click();
            }
        }
        else
        {
            if (string.Equals(p0, "descricao", StringComparison.
                CurrentCultureIgnoreCase))
            {
                if (string.Equals(p1, "ascendente", StringComparison
                    .CurrentCultureIgnoreCase))
```

```
                {
                    selenium.FindElement(By.XPath("//*[@id='
                        F3MGrelhaFormArtigos']/div[1]/div/table/
                        thead/tr[1]/th[3]/a")).Click();
                }
                if (string.Equals(p1, "decrescente",
                     StringComparison.CurrentCultureIgnoreCase))
                {
                    selenium.FindElement(By.XPath("//*[@id='
                        F3MGrelhaFormArtigos']/div[1]/div/table/
                        thead/tr[1]/th[3]/a")).Click();
                    selenium.FindElement(By.XPath("//*[@id='
                        F3MGrelhaFormArtigos']/div[1]/div/table/
                        thead/tr[1]/th[3]/a")).Click();
                }
                selenium.FindElement(By.XPath("//*[@id='
                     F3MGrelhaFormArtigos']/div[1]/div/table/thead/
                     tr[1]/th[1]/a")).Click();
                selenium.FindElement(By.XPath("//*[@id='
                     F3MGrelhaFormArtigos']/div[1]/div/table/thead/
                     tr[1]/th[1]/a")).Click();
            }
            else
            {
                if (string.Equals(p1, "ascendente", StringComparison
                     .CurrentCultureIgnoreCase))
                {
                    selenium.FindElement(By.XPath("//*[@id='
                        F3MGrelhaFormArtigos']/div[1]/div/table/
                        thead/tr[1]/th[2]/a")).Click();
                }
                if (string.Equals(p1, "decrescente",
                     StringComparison.CurrentCultureIgnoreCase))
                {
                    selenium.FindElement(By.XPath("//*[@id='
                        F3MGrelhaFormArtigos']/div[1]/div/table/
                        thead/tr[1]/th[2]/a")).Click();
                    selenium.FindElement(By.XPath("//*[@id='
                        F3MGrelhaFormArtigos']/div[1]/div/table/
                        thead/tr[1]/th[2]/a")).Click();
                }
                selenium.FindElement(By.XPath("//*[@id='
                     F3MGrelhaFormArtigos']/div[1]/div/table/thead/
                     tr[1]/th[1]/a")).Click();
                selenium.FindElement(By.XPath("//*[@id='
                     F3MGrelhaFormArtigos']/div[1]/div/table/thead/
                     tr[1]/th[1]/a")).Click();
            }
        }
    }
```

```
[When(@"seleciona limpar filtros e ordens")]
public void QuandoSelecionaLimparFiltrosEOrdens()
{
    selenium.FindElement(By.XPath("//*[@id='BtRepoe']")).Click()
        ;
}

[Then(@"a pagina dos artigos volta ao estado inicial")]
public void EntaoAPaginaDosArtigosVoltaAoEstadoInicial()
{
    string cCode = selenium.FindElement(By.XPath("//*[@id='
        F3MGrelhaFormArtigos']/div[1]/div/table/thead/tr[1]/th
        [1]")).GetAttribute("data-dir");
    int cDesc = selenium.FindElements(By.XPath("//*[@id='
        F3MGrelhaFormArtigos']/div[1]/div/table/thead/tr[1]/th
        [2]/a/span")).Count;

    if (cCode.Equals("asc") && cDesc < 1)
    {
        Assert.IsTrue(cCode.Equals("asc") && cDesc < 1, "voltou
            ao estado inicial");
    }
    else
    {
        Assert.Fail("ãNo voltou ao estado inicial");
    }
}

[Then(@"'(.*)' deve aparecer nos resultados")]
public void EntaoDeveAparecerNosResultados(string p0)
{
    Thread.Sleep(2000);
    IList<IWebElement> tArt = new List<IWebElement>();
    IWebElement tableRowCollection = selenium.FindElement(By.
        XPath("//*[@id='F3MGrelhaFormArtigos']/div[2]/table/
        tbody"));

    tArt = tableRowCollection.FindElements(By.XPath("//*[@id='
        F3MGrelhaFormArtigos']/div[2]/table/tbody/tr/td"));
    bool isEmpty = !tArt.Any();
    if (isEmpty)
    {
        Assert.Fail("Nao existe");
    }
    else
    {
        foreach (IWebElement row in tArt)
        {
            if (row.Text.Equals(p0))
```

```
            {
                Assert.AreEqual(p0, row.Text);
            }
        }
    }
}

[Then(@"nos resultados devem aparecer todos os artigos")]
public void EntaoNosResultadosDevemAparecerTodosOsArtigos()
{
    string num = selenium.FindElement(By.XPath("//*[@id='
        F3MGrelhaFormArtigos']/div[3]/span")).Text;

    string StateCode = num.Substring(num.Length - 2);
    strJS = "$('#Pesquisa').val('')";
    js.ExecuteScript(strJS);
    selenium.FindElement(By.XPath("//*[@id='Pesquisa']")).
        SendKeys(OpenQA.Selenium.Keys.Enter);
    Thread.Sleep(tempo);
    selenium.FindElement(By.XPath("//*[@id='F3MPaneGrelha']/div
        [1]/span/span/span/span")).Click();
    Thread.Sleep(tempo);
    IList<IWebElement> lVistas = selenium.FindElements(By.XPath(
        "//*[@id='Vistas_listbox']/li"));

    int i = 1;
    foreach (IWebElement row in lVistas)
    {
        if (i == 1)
        {
            row.Click();
            Thread.Sleep(tempo);
            string num2 = selenium.FindElement(By.XPath("//*[@id
                ='F3MGrelhaFormArtigos']/div[3]/span")).Text;
            string StateCode2 = num2.Substring(num2.Length - 2);
            Assert.AreEqual(StateCode, StateCode2);
        }
        i++;
    }
}

[Then(@"o artigo '(.*)' deve desaparecer da tabela")]
public void EntaoOArtigoDeveDesaparecerDaTabela(string p0)
{
    selenium.NavigateTo("http://*****.*****.net:8081/Producao/
        Artigos/Artigos");
    Thread.Sleep(tempo);
    strJS = "$('#Pesquisa').val('" + p0 + "')";
    js.ExecuteScript(strJS);
```

```
        selenium.FindElement(By.XPath("//*[@id='Pesquisa']")).
            SendKeys(OpenQA.Selenium.Keys.Enter);

        Thread.Sleep(tempo);
        IList<IWebElement> tArt = new List<IWebElement>();
        IWebElement tableRowCollection = selenium.FindElement(By.
            XPath("//*[@id='F3MGrelhaFormArtigos']/div[2]/table/
            tbody"));

        tArt = tableRowCollection.FindElements(By.XPath("//*[@id='
            F3MGrelhaFormArtigos']/div[2]/table/tbody/tr/td"));
        foreach (IWebElement row in tArt)
        {
            if (row.Text.Equals(p0))
            {
                Assert.Fail("O artigo ãno foi removido");
            }
        }
    }

    [Then(@"na combo das vistas deve aparecer '(.*)'")]
    public void EntaoNaComboDasVistasDeveAparecer(string p0)
    {
        Thread.Sleep(tempo);

        if (selenium.FindElement(By.XPath("//*[@id='F3MPaneGrelha']/
            div[1]/span/span/input")).GetAttribute("value").Equals(
            p0))
        {
            Assert.AreEqual(p0, selenium.FindElement(By.XPath("//*[
                @id='F3MPaneGrelha']/div[1]/span/span/input")).
                GetAttribute("value"));
        }
        else
        {
            Assert.Fail("A info na combo das vistas ãno é a correta"
                );
        }
    }

    [Then(@"a vista é alterada para o default")]
    public void EntaoAVistaEAlteradaParaODefault()
    {
        string vis = selenium.FindElement(By.XPath("//*[@id='
            F3MPaneGrelha']/div[1]/span/span/input")).GetAttribute(
            "value");

        Thread.Sleep(tempo);
        selenium.FindElement(By.XPath("//*[@id='F3MPaneGrelha']/div
            [1]/span/span/span/span")).Click();
```

```
            if (vis.Equals(selenium.FindElement(By.XPath("//*[@id='
                 Vistas_option_selected']")).Text))
            {
                Assert.AreEqual(selenium.FindElement(By.XPath("//*[@id='
                     Vistas_option_selected']")).Text, vis);
            }
            else
            {
                Assert.Fail("ãNo ficou na vista por defeito");
            }
        }

    }
}
```

A.3 Especificações de baixo nível

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Transactions;
using NUnit.Framework;
using Producao;
using Producao.Repositorio;
using F3M.Base.Cliente.Comunicacao.Modelo;
using F3M.BaseDados.Repositorio;

namespace Testes.Unit
{
    [TestFixture]
    class Artigos_unit
    {
        IRepositorioGenerico<tbArtigos, Artigos> repo = new
             RepositorioArtigos();
        ClsF3MFiltro filtro = new ClsF3MFiltro();

        tbArtigos art = new tbArtigos()
        {
            Codigo = "new_01",
            Descricao = "ardd323tt",
            IDTipoArtigo = 1,
            IDGrupo = 1,
            IDFamilia = 1,
            IDMarca = 2,
```

```
            IDSubFamilia = 1,
            IDUnidade = 2,
            IDUnidadeCompra = 3,
            IDUnidadeVenda = 3,
            UtilizadorCriacao = "F3M",
            DataCriacao = DateTime.Now,
        };

        [Test]
        public void should_add()
        {
            using (TransactionScope scope = new TransactionScope())
            {
                repo.Adiciona(art);
                repo.Grava();

                Artigos art2 = repo.ObtemPorObjID(art.ID);

                Assert.AreEqual(art.Codigo, art2.Codigo);
            }
        }

        [Test]
        public void should_search()
        {
            using (TransactionScope scope = new TransactionScope())
            {
                filtro.FiltroTexto = "novo";

                IEnumerable<Artigos> av = repo.Pesquisa(filtro);
                Assert.AreEqual("novo",av.FirstOrDefault().Codigo);
            }
        }

        [Test]
        public void should_remove()
        {
            using (TransactionScope scope = new TransactionScope())
            {
                repo.Remove(art);
                Assert.AreEqual(null, repo.ObtemPorObjID(art.ID));
            }
        }
    }
}
```