REM

Quality Assurance Engineering

INTERNSHIP REPORT

Rúben Manuel Moreira Santos
MASTERS IN COMPUTER ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

February | 2018

Quality Assurance Engineering

INTERNSHIP REPORT

Rúben Manuel Moreira Santos
MASTERS IN COMPUTER ENGINEERING

SUPERVISOR
Eduardo Leopoldo Fermé

RELATÓRIO DE ESTÁGIO

Quality Assurance Engineering

Relatório de Estágio Submetido à Universidade da Madeira para a

Obtenção do Grau de Mestre em Engenharia Informática

Rúben Manuel Moreira Santos

Funchal, Portugal
Fevereiro 2018

À minha família por toda a motivação

e apoio prestado.

A todos que, de alguma forma, contribuíram.

AGRADECIMENTOS

O objetivo traçado para este estágio foi alcançado graças à contribuição de um vasto numero de pessoas que, direta ou indiretamente, pessoalmente ou profissionalmente, influenciaram de forma positiva todo este período, quer nas minhas decisões como no meu crescimento pessoal e profissional. A todos vós, os meus mais sinceros agradecimentos.

Um especial agradecimento ao meu orientador Professor Doutor Eduardo Fermé, pois foi a pessoa que reconheceu a mais-valia deste estágio curricular e, assim, garantiu que este fosse realizado contribuindo com um precioso apoio ao longo deste período.

Ao Mestre Luís Ferreira, Web Developer da empresa *ABC*, que apesar de não se enquadrar diretamente na minha área de estágio, demonstrou sempre apoio e encontrou-se sempre disponível para me ajudar em tudo o que conseguia, esclarecendo muitas vezes as minhas dúvidas nas partes mais técnicas, permitindo-me assim, desempenhar as minhas tarefas de forma mais eficiente.

À empresa *ABC* por concordar com a realização deste estágio, e a todos os seus colaboradores, por toda a ajuda prestada sempre que esta foi necessária e pelo excelente ambiente que proporcionou uma rápida integração na equipa de trabalho.

Agradeço a todos os meus colegas de curso e amigos, pois estes garantem o equilíbrio emocional que tem um papel tão importante no dia-a-dia.

Para concluir, o meu maior agradecimento aos meus pais e namorada pela paciência demonstrada, garantirem sempre o foco neste projeto e relembrarem sempre a importância desta etapa para o meu futuro, tornando real a concretização dos meus objetivos.

RESUMO

O presente relatório tem como principal objetivo descrever as atividades desenvolvidas durante o estágio profissional, permitindo-me atingir o nível de Mestrado em Engenharia Informática na Universidade da Madeira.

O estágio teve a duração de 1260 Horas e visou a experiência profissional enquanto *quality assurance*, nomeadamente através da participação no processo de desenvolvimento de software destinado a *Smart TV*, Caixas Inteligentes, Dispositivos Móveis e Navegadores Web através da realização de testes sistemáticos às aplicações desenvolvidas.

Foi uma experiência deveras enriquecedora e estimulante para o autor na medida em que lhe permitiu um contato direto com a realidade do desenvolvimento de produtos da Indústria Tecnológica, a melhoria das suas capacidades pessoais e a aquisição de novas habilidades técnicas, bem como, a melhor aprendizagem acerca do campo de testes de software, como estes afetam o ciclo de vida de desenvolvimento e como é importante para garantir a produção de um software de elevada qualidade.

Palavras-Chave:

Aplicação

Teste

Defeito

Software

Quality Assurance

Casos de Teste

Notas de Teste

Desenvolvedor

ABSTRACT

The main objective of this report is to describe the activities developed during the professional internship, which allowed me to reach the level of Master in Computer Engineering at the University of Madeira.

The internship lasted 1260 hours and was based on the professional experiences as a quality assurance, namely through participation in the software development process for Smart TV, Smart Boxes, Mobile Devices and Web Browsers through systematic tests of the applications being developed.

It was a very enriching and stimulating experience for the author because it allowed him a direct contact with the reality of product development in the technological industry, the improvement of his personal capacities and the acquisition of new technical skills, as well as learning about the field of software testing, how it affects the software development lifecycle and how important it is to ensure high quality standard Software.

Keywords:

Application

Test

Bug

Software

Quality Assurance

Test cases

Testing notes

Developeu

ACRÓNIMOS

API - Application Programming Interface

CMS – Content Management System

APP – Application

SVN – Apache Subversion

BUG – Issues within an application

QA – Quality Assurance

DROPS – Process of delivering a version or sub-version of an application

ÍNDICE

INTRODUÇÃO

"A mente que se abre a uma nova ideia, jamais volta ao seu tamanho original"

— Albert Einstein

1. APRESENTAÇÃO

O relatório que se sucede é apresentado no âmbito da conclusão do 2º Ciclo de estudos, enquadrado no curso de Mestrado em Engenharia Informática, pertencente à Universidade da Madeira. Trata-se do fruto de todo o trabalho realizado na empresa *ABC*, resultado da experiência de trabalho adquirida, tornando-se um dos pilares do qual assentou a minha carreira profissional.

"S*cio me nihil scire*", uma frase que se traduz em "Só sei que nada sei". Esta é uma frase que se aplica e enquadra muito bem ao "mundo" do desenvolvimento de software, uma área que se encontra em constante evolução e cujo caminho se encontra repleto de portas abertas para áreas que ainda nos são desconhecidas ou simplesmente pouco exploradas.

Apesar da área da qualidade de software não me ser totalmente desconhecida, considerava como se assim fosse quando iniciei o estágio, pois ainda não é muito explorada pela indústria das Tecnologias de Informação em Portugal. Assim sendo, sinto que encontrei no meu caminho esta porta aberta e decidi agarrar o desafio e a oportunidade de crescer e evoluir dentro desta área.

Acredito que toda esta experiência serviu como uma preparação para poder progredir em termos técnicos dentro da imensidão que é a área de qualidade de software, assim como, melhorar o meu trabalho em equipa em prol de um objetivo comum.

1.1.CONTEXTO DO ESTÁGIO

Estando previsto, pelo plano de estudos do Mestrado em Engenharia Informática, enquadrado na Unidade Curricular "Dissertação, Trabalho de Projeto ou Estágio", surgiu, então, a curiosidade de querer experimentar, em primeira mão, desempenhar funções enquadradas na minha área de estudos, de modo a por à prova o conhecimento adquirido a nível académico, melhora este conhecimento e melhorar a nível profissional.

Após um período bastante frustrante e desanimador, no qual tentei realizar uma Dissertação de Mestrado, mas, sem sucesso, deparei-me com a necessidade de procurar algo diferente.

Surge, através de um amigo, a oportunidade para realizar um estágio relacionado com desenvolvimento de software, desempenhando funções de *quality assurance* na parte da qualidade de software, para uma empresa sediada em Londres, mas que, contemplava na sua estrutura laboral, um escritório situado no centro do Funchal.

Alguns meses após iniciar esta aventura a desempenhar funções na área da qualidade de software, apercebi-me do quão fascinante estava a ser "trabalhar" numa área que me era, em grande parte, desconhecida e, após alguma reflexão, resolvi avançar para a realização de estágio, o qual foi aceite por parte da empresa, visto que ia de acordo com as minhas tarefas desempenhadas.

Um dos fatores que também me direcionou para a realização de estágio, foi por considerar que os estágios são melhores alternativas a dissertações, pelo facto de o aluno ter a oportunidade de estagiar num ambiente sem ser o académico, testar os seus conhecimentos, poder, a partir desse estágio já estabelecer um cargo de trabalho ou contactos para esse efeito e experimentar o mercado de trabalho de forma a se preparar para o que terá que enfrentar.

1.2.CONTEXTO INSTITUCIONAL

A organização selecionada para a realização deste estágio, como abordado previamente, surge através de uma oferta de emprego, a qual foi aceite pela minha parte, e mais tarde, por iniciativa própria, resolvi propor a realização de estágio, enquadrado no meu cargo dentro da empresa, que obteve resposta positiva por parte da direção.

Por já não me encontrar a colaborar profissionalmente com a empresa, passarei a referir-me à mesma como *ABC*, sendo esta uma empresa relativamente recente, com cerca de 5 anos, que opera maioritariamente a nível do desenvolvimento de software para *Smart Tv's*, tendo vindo a crescer também em outras áreas, nomeadamente, na criação de aplicações para dispositivos moveis com bastante sucesso.

A evolução da empresa e o crescimento da mesma, obrigou à sua expansão, tanto a nível internacional como em termos do seu escritório aqui no Funchal, a nível de espaço e a nível laboral. Devido à falta de experiência no exercício de funções enquanto *software developer* e à falta de trabalhadores qualificados em *quality assurance* em Portugal, abriu-se-me as portas para esta oportunidade, conectando o meu historial académico a área das tecnologias de informação.

1.2.1. Descrição da Empresa

A empresa ABC, legalmente formalizada como ABC, Unipessoal LTD, é uma empresa com foco no desenvolvimento de software direcionado para a distribuição de conteúdo multimédia ou conteúdo OTT/VOD (*over-the-top content / vídeo-on-demand*).

Este conteúdo é desenvolvido para ser distribuído por múltiplas plataformas simultaneamente, abrangendo websites, dispositivos móveis (*smartphone, tablet, iphone, ipad*), *Smart TV's, Smart Boxes* e Sistemas de Jogo (*Xbox e Playstation*).

De forma a garantir toda esta distribuição numa larga escala de conteúdo, a empresa conta também com o seu próprio sistema online de gestão de conteúdo, sendo desenvolvido um sistema de gestão de conteúdo para cada aplicação desenvolvida, permitindo ao cliente, ter controlo total sobre todo o conteúdo que exista na aplicação e, permitindo fazer mudanças, visíveis em tempo real. Este sistema também permite a integração de outros serviços que providenciam conteúdo multimédia, fazendo a gestão deste conteúdo recebido.

1.2.2. Localização

A empresa ABC possui instalações espalhadas pela Europa, apresentando, nesta fase, um foco maior de recursos em Portugal com um escritório localizado no centro do Funchal, e outro localizado na Covilhã, sendo que, a sede situa-se em Londres, local de nascimento da empresa. Para além destes escritórios, a empresa apresenta também um funcionário na Flórida, encarregue de tratar de vendas dos produtos ABC em território americano. Conta também com vários funcionários na sede, em Londres e com um funcionário localizado em Barcelona, Espanha.

1.2.3. Estrutura Interna e Organograma

A estrutura interna da empresa ABC está organizada nas seguintes direções:

- Direção
- Departamento de Produção
- Departamento de Vendas
- Direção de Marketing;

Cabe à Direção, a responsabilidade de planeamento e estratégia da empresa ABC. A Direção de Produção, desenvolve um produto ou sistema. A Direção de Vendas e a Direção de Marketing têm como objetivos a aquisição de contactos, a relação com o cliente e estratégia de marketing para divulgação dos produtos e serviços que a empresa contempla.

A Direção, para além de ter a responsabilidade financeira e o planeamento das estratégias da empresa, também contempla responsabilidades nos departamentos de marketing, vendas e produção, sendo que, ambos os diretores participam ativamente no departamento de vendas, e, um deles participa também ativamente nos departamentos de Produção (estando encarregue de gerir os servidores da empresa ABC e desempenhando também funções de Project Manager) e também no departamento de Marketing (tomando decisões para novas estratégias de marketing).

A estrutura pode ser visualizada no seguinte organograma:

Figura 1 - Organograma simplificado da empresa ABC

No Anexo 1 é possível visualizar uma versão do organigrama com mais detalhe.

1.3. TAREFAS A DESEMPENHAR DURANTE O ESTÁGIO

Durante o período no qual decorreu este estágio, a empresa *ABC* colocou ao meu dispor, um espaço de trabalho de forma a cumprir o horário estabelecido, assim como material (computador portátil e um rato de computador) que me permitisse desempenhar as minhas funções como *quality assurance*. Para além disto, fui acompanhado durante o primeiro mês por uma colega que também desempenha funções de *quality assurance*, de forma a aprender os vários processos necessários para a realização dos testes, e esta manteve-se sempre disponível ao longo de todo o período de estágio para me ajudar e esclarecer possíveis dúvidas que pudessem surgir.

No primeiro mês, que considero como o período de adaptação, comecei por ter acesso às várias ferramentas utilizadas pela empresa, de forma a criar as respetivas contas necessárias para a utilização das mesmas.

Fui introduzido aos vários procedimentos de teste, assim como às várias ferramentas e plataformas, o que acabou por ser um processo bastante rápido e simples, permitindo que o meu contributo para a equipa pudesse ser feito mais rapidamente.

Desde da fase de entrevista, que o objetivo traçado era bastante claro, encontrar sempre o maior número possível de bugs nas aplicações, tornando-se óbvio qual teria que ser o foco no dia-a-dia.

Através de conversas com os restantes colegas, consegui fazer um levantamento de informação sobre quais os bugs que estavam a acontecer com mais frequência, quais os projetos com mais bugs e quais as plataformas de testes que normalmente apresentavam maior numero de bugs por projeto, permitindo-me assim organizar melhor o meu tempo de forma a poder realizar testes mais rigorosos e com maior atenção quando as tarefas a serem realizadas envolviam estes projetos/plataformas.

2. ESTADO DE ARTE

*2.1.*ORIGENS DE *QUALITY ASSURANCE*

O controlo de qualidade consegue ser datado até ao tempo medieval da história da humanidade, onde o controlo de qualidade era realizado por artesãos que se juntaram cooperações (*guilds*) e criaram uma espécie de produção industrializada para a altura.
Este modelo foi seguido até ao inicio do século 19 por toda a produção industrializada a nível mundial.

No inicio do século 20, é que os fabricantes começaram a incorporar processos de controlo e práticas de qualidade na produção dos seus artigos.

Durante as décadas de 60, 70 e 80, foi um período um pouco negro para o desenvolvimento de software, no qual, foram identificadas muitas das falhas existentes no processo de desenvolvimento de software, desde projetos a ultrapassar o orçamento previsto e o tempo de desenvolvimento, e, em casos extremos, danos de propriedades e até perdas de vida. [1]

Focando agora de forma mais especifica, a história de *quality assurance* no software, nas primeiras décadas da computação, todo o desenvolvimento era feito numa abordagem *ad hoc*, na qual o software era desenvolvido para um fim especifico. Neste tipo de abordagem, *quality assurance* também seguia a mesma metodologia. Nesta altura, o desenvolvimento de software consistia em projetos pequenos, equipas com poucos membros, software relativamente simples e sem grande pressão para serem feitas entregas regularmente, todo o processo de *quality assurance* era feito como parte de um amplo processo de *debugging*.

Avançando para as décadas de 1990 e 2000, com a produção em massa de computadores para uso doméstico e a variedade de computadores existentes no mercado, significava que todo o software desenvolvido teria que ser compatível com todos os tipos de computadores existente de forma a ter sucesso, o que elevou a fasquia

para *quality assurance*, levando a otimizar os processos de *quality assurance* dentro das empresas, e evoluindo esta área até há atualidade. [2]

2.2.MÉTODOS DE *QUALITY ASSURANCE*

Os métodos de *quality assurance*, podem ser divididos, de uma forma geral, em duas categorias: *BlackBox testing* e *WhiteBox testi*ng, os quais irei abordar de seguida.

BlackBox testing (teste Caixa Preta) - refere-se a testar a parte funcional ou interface, como é normalmente referida, de um software. Neste tipo de testes, apenas é assegurado que tudo funciona como o esperado, independentemente de ter havido algum tipo de alteração no software a nível de *backend*, desde que o software continue a fazer aquilo que é suposto fazer, este é aprovado pelo *quality assurance*. [3]

Este tipo de testes, é muito utilizado em empresas que têm o departamento de testes como um departamento separado do processo de desenvolvimento, em que o departamento de testes apenas participa exclusivamente no processo de desenvolvimento na fase testes, ou então, em empresas nas quais os testes têm dificuldade em perceber código de programação ou não têm experiencia nesta área.

BlackBox testing contempla os seguintes tipos de teste:

1. *System Testing* – é testado o software na sua totalidade. Aqui o objetivo é verificar condições de trabalho esperadas pelo utilizador, assim como verificar possíveis condições de exceções ou condições extremas para o software. [4]

2. *Acceptance testing* – o software é verificado para garantir que todos os requisitos do projeto foram implementados e que o software foi aprovado pela equipa de *Quality Assurance* e pelo cliente de forma a garantir que este funciona de acordo com o esperado e que cumpre os requisitos especificados pelo cliente aquando do inicio do projeto. [4]

3. *Functional testing* – o componente é testado de acordo com as especificações do software ou de acordo com a descrição de como o software deve funcionar. O software é testado através de ações que resultam em inputs ao componente, e é observado o correspondente output por parte do

componente, de forma a verificar se este é o esperado ou não. Por sua vez, o teste funcional aborda 4 tipos de testes diferentes, sendo eles:

- *Smoke testing* – trata-se de um teste focado apenas nas principais funções do software, de forma a verificar se estas apresentam alguma falha critica que possa fazer com que o software seja rejeitado. [4]
- *Sanity testing* – é realizado um teste rápido e geral de todo o software de forma a garantir que não existe erros críticos e que os principais cenários de execução estão a ser executado corretamente. [4]
- *Regression testing* – o software é testado depois de alguma alteração ter sido feita ou um novo componente ter sido integrado, de forma a garantir que todos os componentes previamente desenvolvidos continuam a funcionar corretamente. [4]
- *Usability testing* – o software é testado por utilizadores de forma a poder ser avaliada a usabilidade do software, garantindo que o utilizador consegue facilmente entender qual o seu objetivo e consegue facilmente realizar ações no mesmo. [4]

Os testes mencionados anteriormente, são, de forma simples, todos referentes a testar o software e as suas funções, sendo uns de forma mais geral, e outros, de forma mais minuciosa.

Estes testes mencionados, eram os mais praticados pela equipa de QA, nos quais, era simulado, da melhor maneira possível, que a aplicação era usada por um utilizador real sem conhecimento nenhum sobre a aplicação, como, por exemplo, realizar tarefas simples como abrir um determinado menu, ou tarefas mais complexas, como, por exemplo, efetuar um login dentro da aplicação.

WhiteBox testing (teste Caixa Branca) – refere-se a testar o "interior" do software ou *backend* do software. Neste tipo de testes, é verificado, por exemplo, se o software lança algum tipo de exceções em determinadas condições de execução, então este pode ser testado nessas mesmas condições para determinar o impacto dessas exceções no

software. Estes testes, requerem que os *QA's* tenham conhecimento a nível de programação e conhecimento interno do software. [3]

WhiteBox testing contempla os seguintes tipos de teste:

1. *Unit testing* – o software é desenvolvido em módulos que vão sendo anexados sucessivamente até termos a versão final do software. Este tipo de testes, testa cada módulo do software individualmente de forma a verificar que realiza todas as funções que foram previamente definidas. [4]

2. *Integration Testing* – após a verificação dos módulos a nível individual, é necessário verificar que estes conseguem cumprir as suas funções, livres de bugs, quando acoplados com outros módulos constituintes do mesmo software e que necessitam de comunicar de forma a desempenhar determinadas tarefas. Nesta fase de testes, os módulos já desenvolvidos, são anexados aos módulos que partilham alguma tarefa ou que são diretamente dependentes um do outro e são testados como sendo apenas um só módulo. [4]

Os testes mencionados não eram efetuados pela equipa de QA, pelo facto de a equipa, maioritariamente, não ter conhecimento a nível de programação e pelo facto de também não ser concedido acesso ao código da aplicação para fins de teste.

2.3.METODOLOGIA AGILE

Os métodos ágeis surgiram através da experiencia pessoal e profissional de desenvolvedores de software que viveram em 1ª mão as limitações e frustrações que os métodos de desenvolvimento de software tradicionais criavam nos projetos desenvolvidos, e que, projeto após projeto, continuavam a aparecer e atormentar os desenvolvedores. A abordagem dos métodos ágeis vem combater diretamente os problemas que os métodos tradicionais apresentam, quer a um nível geral como a um nível mais especifico de desenvolvimento.

Na sua forma mais pura, desenvolvimento ágil ou *agile development*, é simplesmente um "livro" com instruções bastante simples que permitem às equipas evoluírem constantemente, ao longo do desenvolvimento do projeto, a nível técnico e funcional, mantendo sempre o foco na entrega rápida de valor comercial. Como resultado, as empresas conseguem apresentar, em geral, um risco de nível baixo associado ao desenvolvimento de software.

Através destes métodos, é possível acelerar a entrega inicial de valor comercial, e como estes métodos são dinâmicos, estes baseiam-se num processo continuo de planeamento e feedback, que permite maximizar o valor comercial ao longo de todo o processo de desenvolvimento. [5]

Com todo este processo iterativo de planeamento e feedback constante entre a equipa e entre várias equipas, é possível, às equipas envolvidas, fazer alterações ao software de acordo com as necessidades de negocio e fazer uma rápida alteração do software para que este vá sempre de acordo com os requisitos desejados.

O ciclo de vida do desenvolvimento de software em metodologias *agile*, contempla as seguintes fases:

I. Planeamento – o processo começa o planeamento de todo o software, em que irá consistir, qual o(s) objetivo(s) a cumprir, o que será mais prioritário para ser desenvolvido. É também estimado o tempo de desenvolvimento para as várias entregas do software.

II. Análise de requisitos - É feita uma análise de requisitos especificados para o software em questão, de forma, a que nesta fase seja já feita uma análise de possíveis falhas que o software poderá encontrar na fase de implementação e assim definir já soluções para essas situações.

III. Design – é feito todo o design do software

IV. Implementação - é iniciada a fase de implementação

V. Fase de testes

   A. *Testing* – são realizados vários testes diferentes de forma a testar vários aspetos do software.

   B. Testes de aceitação – são realizados testes que permitem determinar se a implementação está correta de acordo com os requisitos do software e se esta pode ser entregue ao cliente.

VI. Manutenção – prestar apoio a qualquer problema que possa aparecer relacionado com a aplicação entregue ao cliente.

Na figura 2, podemos observar o ciclo de desenvolvimento de software

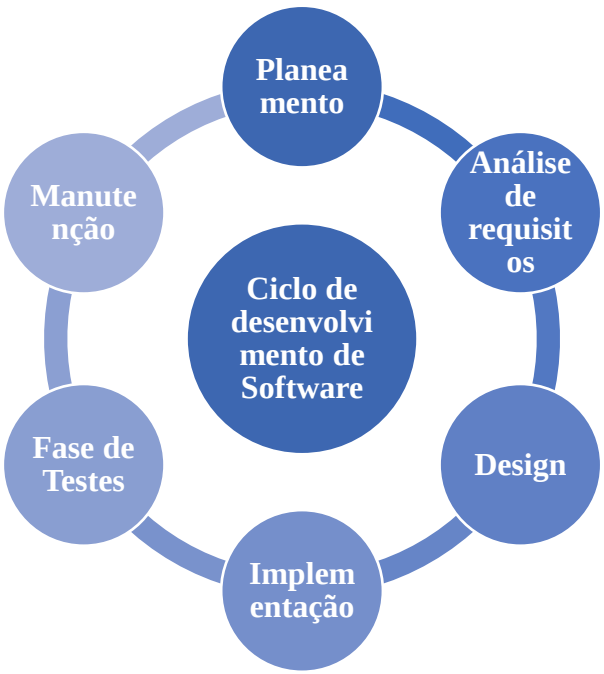


Figura 2 – Esquema do Ciclo de Desenvolvimento de Software

2.3.1. Metodologia Agile em *Quality Assurance*

Para falarmos em desenvolvimento AGILE, devemos primeiro falar, de forma sucinta, sobre os métodos de desenvolvimento tradicionais.

Estes, baseavam-se maioritariamente em metodologias em cascata (*waterfall*) como processo de desenvolvimento, nas quais, o desenvolvimento era todo realizado, e, apenas numa fase final, eram realizados todos os testes (*quality assurance*) necessários. A figura 3 mostra, sobre a forma de um diagrama, como funcionava esta metodologia:

Figura 3 – Diagrama da Metodologia de Desenvolvimento de Software baseada em Cascata.

Isto criava a necessidade de existir imensa documentação, de forma a garantir que os testes criados iam de acordo com que o que tinha sido especificado para o software. [7, 10]

Com este tipo de metodologia, advém vários problemas:

- Excesso de redundância no esclarecimento de requisitos [8, 10]
- Imensas oportunidades para gerar falta ou falha de comunicação [8, 10]
- Quando um erro é descoberto, haverá um atraso significativo até que este seja resolvido [8, 10]

De forma a melhorar todo o processo de desenvolvimento, *quality assurance* foi movido para mais perto da fase inicial do desenvolvimento, permitindo reduzir redundância, falhas ou falta de comunicação e atrasos na resolução de problemas encontrados, assim como a realização de testes ao longo de todo o desenvolvimento, ao invés de estar restringido a apenas uma fase final de testes.

Com esta mudança, *quality assurance* passou a ter um papel importante em todo o processo de desenvolvimento de software, mas quão importante será esse papel ou qual o impacto que realmente tem? Percebendo, em cada fase de desenvolvimento de metodologia *agile*, aquilo que *quality assurance* pode alcançar e qual o efeito que este terá no processo de desenvolvimento, serão as respostas à pergunta colocada, pelo que, passarei a falar de forma mais pormenorizada, o papel de *quality assurance* nas diferentes fases do processo de desenvolvimento de software com metodologia *agile*. [6, 7, 9]

Como referido anteriormente neste relatório, o ciclo de vida *agile* é composto por 6 fases: Planeamento, Análise de requisitos, Design, Implementação, Fase de Testes e Manutenção. Em cada uma destas fases, podemos ter *quality assurance* e iremos agora abordar como é que isso pode ser alcançado:

Numa fase de planeamento, *quality assurance* consegue, desde logo, influenciar o fluxo de todo o planeamento através de algumas perguntas como: "Qual o volume de desenvolvimento e de testes que estamos a considerar obter ao longo de todo o desenvolvimento? O que vamos usar como métricas para a realização dos testes necessários e avaliar o sucesso de todo o projeto? Qual será o processo de *deploy* e de testes? Qual será o processo de testes a cada fase? Todas estas questões, permitem, estabelecer desde do inicio princípios a serem seguidos ao longo do desenvolvimento e permite que a equipa tenha um guia de como proceder a cada fase, assim como, esclarecer logo questões que numa fase mais avançada poderiam criar atrasos no desenvolvimento. Nesta fase, também permite ao *quality assurance* obter uma visão global de todo o projeto que será desenvolvido, o que o ajudará na criação dos seus testes, e, caso exista *quality assurance* manual e de automação, obter também uma visão do que deverá ser verificado pelo QA manual e pelo QA de automação.

Durante a análise de requisitos, o *quality assurance* pode fazer um refinamento destes, tentando extrair o máximo de informação, de forma a conseguir estabelecer quais os critérios que cada requisito deve cumprir, de forma a que estes fiquem bem declarados, e assim, criar condições favoráveis para que os testes criados consigam cobrir a 100% todas as condições impostas, e que os *developers* tenham clareza sobre o que devem implementar e qual o comportamento que cada componente desenvolvido deve ter. Tudo isto também ajuda a prevenir redundância de comunicação por parte dos *developers*, não existindo necessidade de pedidos de informação constante sobre o que desenvolver, qual o comportamento, o que deve e não deve ser possível fazer.

Ao longo de todo o design, *quality assurance* pode verificar se o *design* implementado pode, de alguma forma, criar algum tipo de conflito com os requisitos estabelecidos ou com o comportamento pretendido para estes. Também permite realizar um refinamento do comportamento de cada requisito e, desta forma, estabelecer ainda mais definição sobre toda a informação existente em torno dos requisitos e dos seus comportamentos.

Durante a fase de implementação, os *developers* iniciam a implementação dos vários componentes do projeto, e, ao mesmo tempo, os *quality assurance* iniciam a criação de *test cases ou testing notes* (no caso de *quality assurance manual*) e dos vários cenários (por exemplo criação/inserção/alteração de dados em bases de dados, cenários de validação de dados) no caso de *quality assurance de automação*. (é possível ver um exemplo de *testing notes* na figura 11).

EXECUTION STEPS	EXPECTED RESULTS	ACTUAL RESULT	DATA USED
Click the OK button to clear the error.	Error message is no longer displayed	as expected	
Clear all fields, then in the "Last Name" field, enter a letter and click on the "Send" button.	Error message appears listing the fields that are required and not yet populated	as expected	A
Click the OK button to clear the error.	Error message is no longer displayed	as expected	
Clear all fields, then in the "Email Address" field, enter a letter and click on the "Send" button.	Error message appears listing the fields that are required and not yet populated	as expected	A
Click the OK button to clear the error.	Error message is no longer displayed		
Attempt to enter more than five characters in the "Postal Code" field.	Only the first five are accepted	FAIL - no validation for standard length?	59827387
Attempt to enter less than five characters in the "Postal Code" field.	Validation on leaving?	FAIL - no validation for standard length?	222
Clear all fields, then in the "Postal Code" field, enter the 36746 and click on the "Send" button.	Error message appears listing the fields that are required and not yet populated	as expected	36746
Click the OK button to clear the error.	Error message is no longer displayed		
Click on the "Cancel" button.	Main MOP page is displayed	as expected	
Click on the "Click Here" link in the "Lost Your Password" section at right of the home page.	Forgot Password page is displayed	as expected	
Enter data in the following required fields: First Name, Last Name, Email Address. Verify that registration is allowed.	Registration Success message displays	as expected	First Name: Mildred Last Name: Ames Email Address: PSoftTester01@gmail.com

Figura 11 - Um pequeno exemplo de *testing notes*, criadas para a realização de testes. Estas ajudam a garantir que todos os cenários possíveis para o componente a ser testado, são verificados.

Caso exista possibilidade, é bastante vantajoso que a equipa de QA tenha possibilidade de efetuar verificações contínuas em tempo real ao longo do desenvolvimento, permitindo uma rápida troca de informação entre *QA's* e *developers* sobre possíveis *bugs* ou falhas que estejam a ocorrer, e também, possíveis comportamentos que não estejam corretos ou que estejam a suscitar algum tipo de dúvida. Aqui, *quality assurance* intervém evitando ao máximo, que sejam provocadas demoras no desenvolvimento e entrega dos vários componentes do projeto.

Numa fase de testes, são executados, casos existam, os testes de automação previamente criados e definidos, assim como, todos os testes manuais previamente definidos como necessários para a validação de todos os componentes e, do projeto como um todo, com todos os componentes a funcionar em conjunto.

Nesta fase, o mais vantajoso e claro, se possível, seria que todos os testes fossem primeiro executados num ambiente de controlo (ambiente de teste) que, replicasse, o mais aproximadamente possível, o ambiente final no qual o produto final será lançado, de forma a obter um feedback inicial do impacto do ambiente no produto e, caso existam, identificar em antemão erros críticos que possam comprometer o bom funcionamento e qualidade do produto, assim como, decidir se pequenos *bugs* ou falhas que sejam identificados, devem ser corrigidos antes do produto passar para o ambiente final, ou se estes podem ser corrigidos a seu tempo. Mais uma vez, *quality assurance* ajuda na prevenção de atrasos significativos para o projeto, assim como, na tomada de decisões que podem influenciar diretamente o cumprimento dos prazos estabelecidos.

A fase de manutenção, trata-se de corrigir pequenos *bugs* ou falhas que foram acordados para uma fase pós-término do projeto, e, também de melhoramento dos vários componentes desenvolvidos, reiniciando todo este ciclo que foi descrito até agora. [6, 7, 10]

2.4. FERRAMENTAS DE APOIO

Em cada tipo de testes, existem ferramentas que permitem auxiliar as tarefas a serem executadas pelo *quality assurance*. Abordando cada tipo de teste anteriormente mencionado, irei agora fazer referencia a algumas ferramentas existentes e falar sobre estas de forma geral.

- *System Testing* – de forma a testarmos o software na sua totalidade de forma mais rápida, podemos utilizar ferramentas como *Selenium, Fit*, *WET* ou *Watir*. Todas estas são ferramentas de automação de testes, que permitem testar todo o software rapidamente, assim como verificar infinitas situações de utilização do software. Este tipo de ferramentas será abordado novamente no decorrer deste relatório. [11]
- *Acceptance testing* – novamente, temos ferramentas de automação de testes (*Fit*) como auxilio. Sendo que, neste tipo de testes, é verificado se o software cumpre os requisitos especificados, torna-se mais simples e rápido automatizar esta validação. [11]
- *Functional testing* – mais uma vez, ferramentas de automação de testes permite auxiliar neste tipo de testes, assim como, os sub-testes associados. Como, de forma geral, se tratam de testes para fazer verificações não muito detalhadas do software e fazer verificações simples de deteção de erros, ferramentas de automação permitem a rápida realização deste tipo de testes. [11]
- *Unit testing* – para serem testados componentes, temos ferramentas como *JUnit*. Esta ferramenta permite testar partes do código implementado para um determinado componente ou parte dele, de forma a verificar um determinado comportamento ou estado, mas limitam-se a componentes de simples interação. [11]
- *Integration testing* – como nestes testes é verificado o correto funcionamento entre um novo componente com os restantes anteriormente implementados, necessitámos de ferramentas que consigam suportar um tipo

de complexidade superior. Para tal, existem ferramentas como *HttpUnit*, *SoapUI*, que, permitem verificar ações mais complexas (como submissão de formulários, validações *JavaScript*, autentificação) que podem ser efetuadas entre os diversos componentes do software, garantindo assim que essas ações são validadas para um correto funcionamento do software. [11]

Das diversas ferramentas de apoio referidas, não existe maneira concreta de dizer que a ferramenta A é a melhor ou que a ferramenta B é a mais completa, tratando apenas de uma questão de optar pela ferramenta que melhor se adapta às tarefas que se pretende realizar. De acordo com o critério mencionado, as ferramentas referidas anteriormente são apenas algumas das ferramentas mais populares, e, mais usadas pela comunidade de *quality assurance*.

Vejamos agora, as várias ferramentas mencionadas, com maior detalhe:

- *Selenium* – a principal função desta ferramenta, é automatizar testes em *web browsers*, ou, por outras palavras, simular ações que um utilizador pode realizar ao utilizar um *web browser* (como clicar, inserir dados, apagar, entre outros). Para tal, esta ferramenta fornece duas formas de conseguir isto: através de um ambiente de desenvolvimento próprio – *Selenium IDE* (podemos ver um exemplo disso na figura 4), ou através da função *Record and Play (*Gravar e Reproduzir*) (*visível na figura 5). A primeira opção, requer conhecimentos de programação por parte do utilizador, mas como resultado final, obtemos testes mais robustos que podem ser distribuídos por vários ambientes. A segunda opção não requer conhecimentos de programação por parte do utilizador, pois este fornece uma interface para que se inicialize todo o ambiente, que depois de iniciado, o utilizador apenas precisa de utilizador o *web browser* como normalmente faria, que todas as ações efetuadas serão gravadas, sendo, apenas necessário, terminar o *software* quando se der por terminada a utilização. Após esta primeira utilização, apenas é necessário, executar o ficheiro que gravou todas as ações efetuadas, para que o *software*, repita de forma automática, todas essas ações, fornecendo respostas de sucesso ou insucesso a cada ação, de forma

a validar cada uma delas. [12]

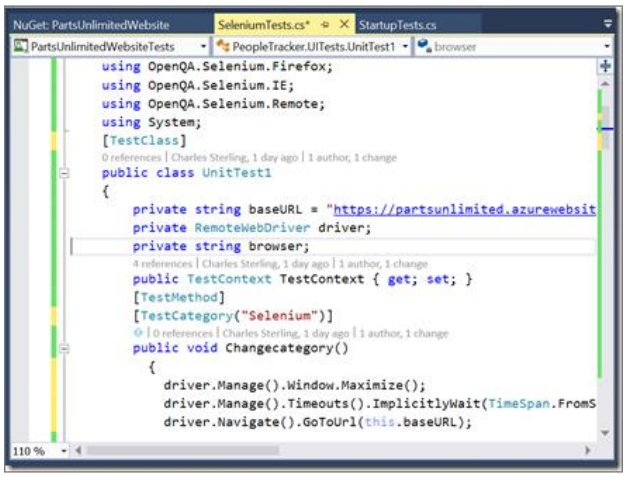


Figura 4 – exemplo de um *script* de automação utilizando o *Selenium IDE*

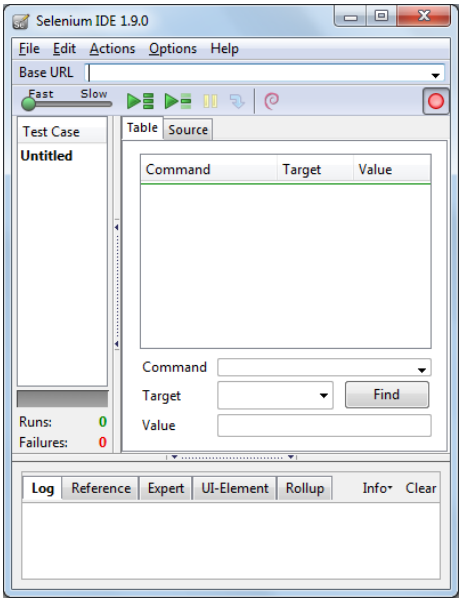
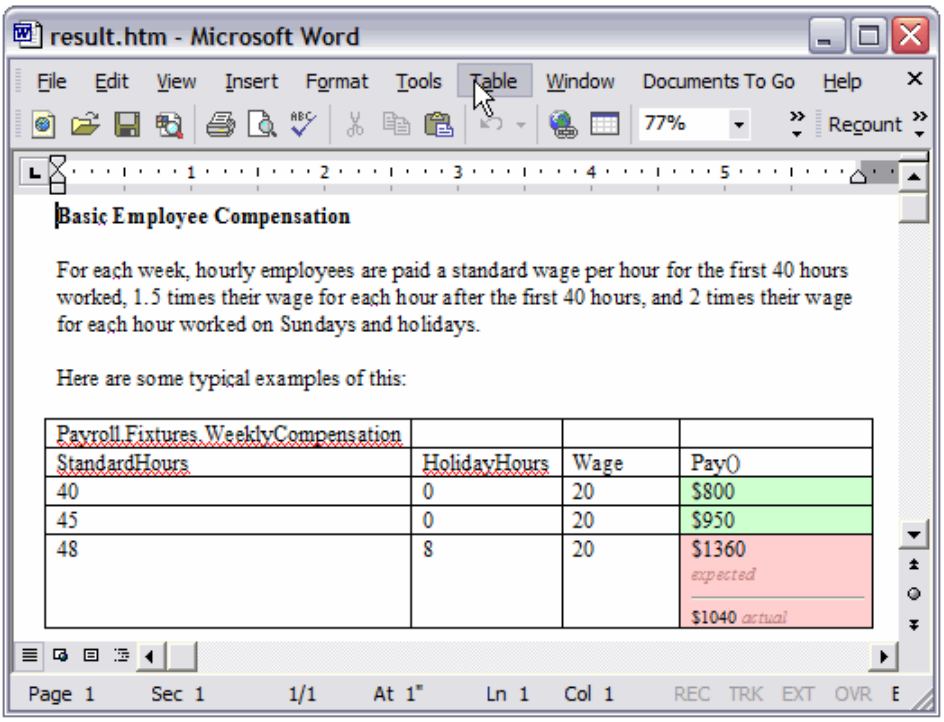


Figura 5 – exemplo da interface da opção *record and play* da ferramenta *Selenium*.

- *Fit (framwork for integrated test)* – uma ferramenta criada para auxilio na melhoria da comunicação ao longo do processo de desenvolvimento. Esta permite que, tanto a equipa como os clientes, aprendam, o que o *software* deveria fazer e aquilo que realmente está a fazer. Este funciona, através da criação de tabelas que contêm exemplos relativos ao funcionamento esperado do *software*, que depois são gravadas em ficheiros com formato *HTML* (podemos ver um exemplo das tabelas utilizadas na figura 6). Cada coluna da tabela, é depois, associada a uma classe existente no programa, que depois utiliza uma função própria do *Fit* para comparar os exemplos existentes nas tabelas, com os

resultados obtidos ao executar o programa com os valores dos exemplos fornecidos. No final de cada exemplo executado, a coluna com a ação final de cada exemplo, é preenchida com um fundo vermelho ou verde, de acordo com a validação ou rejeição do teste. [13, 14]



A Fit document showing success (green) and failure (red)

Figura 6 - Um exemplo de um ficheiro da ferramenta *Fit*, no qual podemos verificar os resultados de um teste que foi executado.

- *Wet (Web Tester)* e *Watir* – assim como o *Selenium*, estas ferramentas de automação, exigem, por parte do utilizador, conhecimento a nível de programação para implementação dos testes, e funcionam de forma semelhante à primeira opção mencionada na ferramenta *Selenium*. O objetivo, é, mais uma vez, o de simular ações que um utilizador pode realizar de forma automática, com validação das várias ações. [15, 16]

- *JUnit* – temos aqui, mais uma *framework* de desenvolvimento de testes de automação, mas, com a particularidade de, permitir testar cada componente do

software de forma isolada, sem existir a obrigatoriedade de testar todo o *software* a cada ciclo de testes (é possível ver um exemplo da interface da ferramenta JUnit e resultados da execução de testes na figura 7). [17]

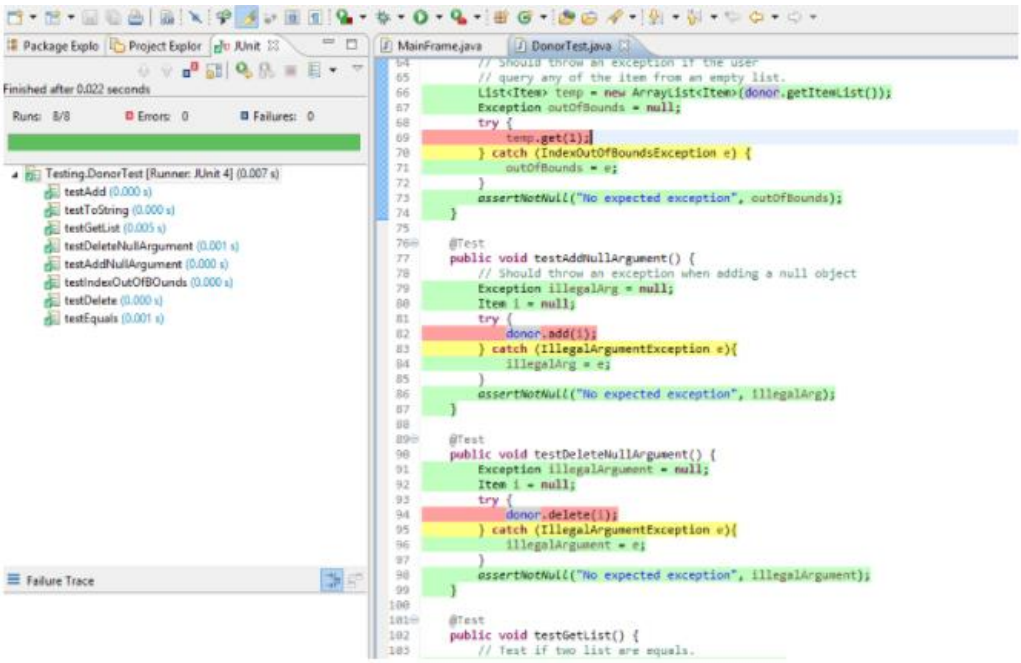


Figura 7 – Exemplo de um teste e consequentes resultados através da ferramenta *JUnit*.

- *HttpUnit* – trata-se de uma *framework* de desenvolvimento de testes automáticos que simula várias funções existentes num *web brower* (como submissão de formulários, autentificação, entre outros), mas, retira a necessidade de se ter um *web browser* instalada no computador. [18]

- *SoapUI* – uma ferramenta que permite executar variados testes a *web browsers*. Funciona de forma semelhante ao método *Record and Play* do *Selenium* (exemplos desta funcionalidade nas figuras 8 e 9), na qual o utilizador não precisa de conhecimentos de programação, mas é obrigado a definir os testes pretendidos, antes de os executar. Isto é tudo feito através da interface gráfica da ferramenta, através de ações de arrastar e largar. [19]

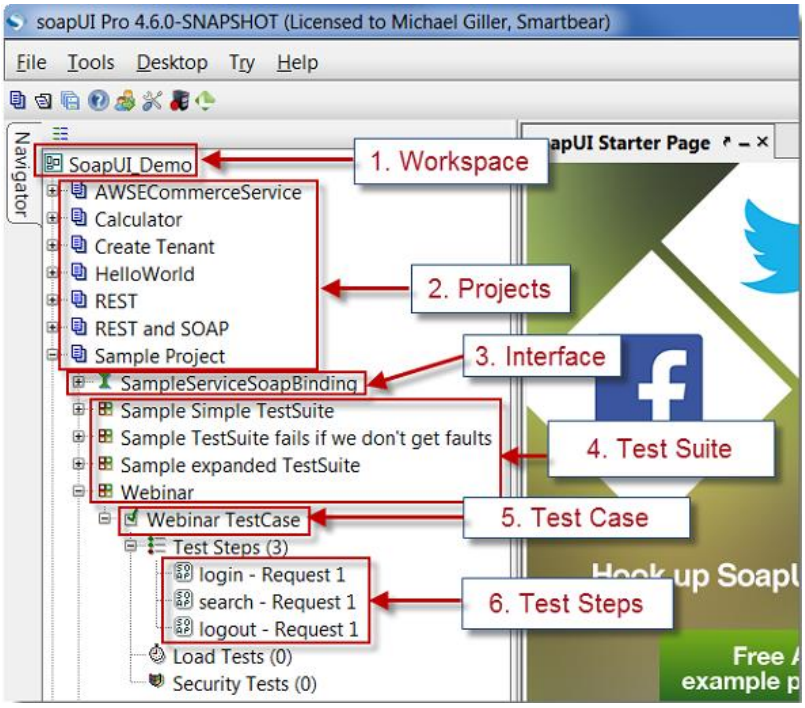


Figura 8 - Temos um uma visão geral de como o projeto fica organizado, e as opções que podemos "pegar" e "arrastar".

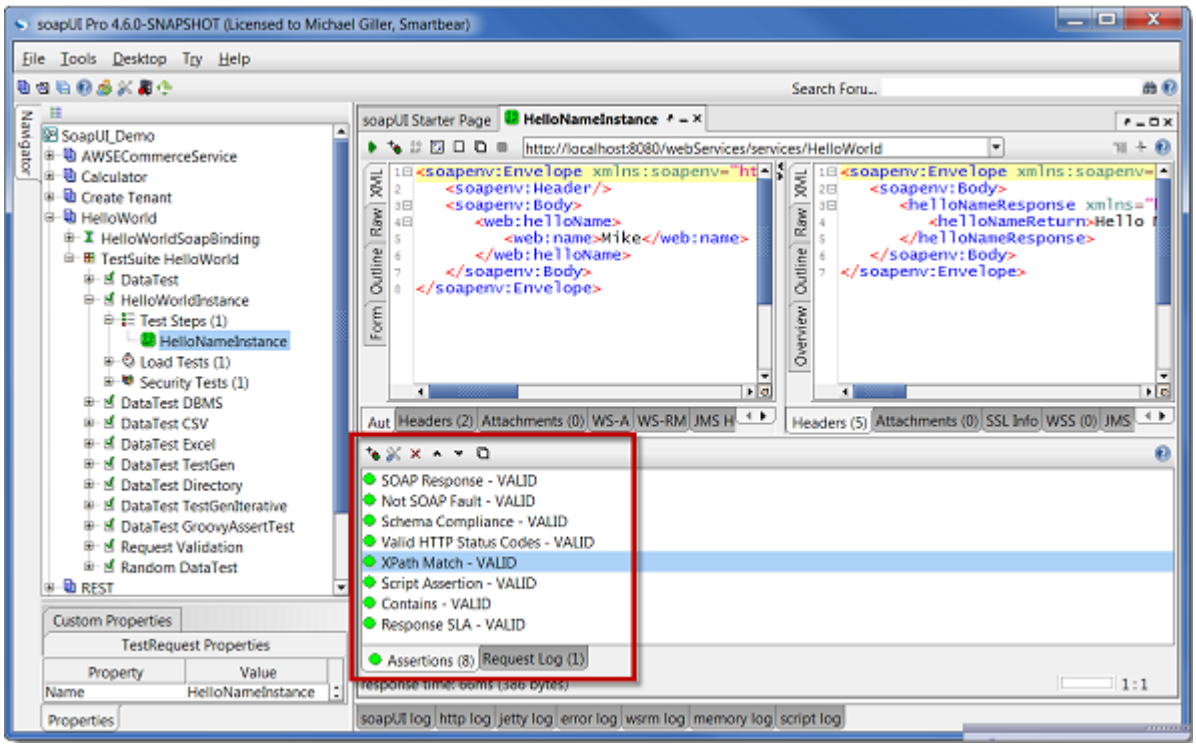


Figura 9 - Nesta imagem, temos o exemplo de um teste com as acções pretendidas já definidas e alguns resultados dessas acções.

Uma grande vantagem, em geral, destas ferramentas, é tratarem-se de ferramentas *open source* e existir uma grande comunidade a nível global, que utiliza diariamente estas ferramentas, existindo um enorme nível de suporte e documentação quando deparados com alguma dificuldade. Outra grande vantagem que encontrámos neste leque de ferramentas, é o fato de os utilizadores não serem obrigados a aprender uma nova linguagem de programação, caso o utilizador já tenha experiencia numa determinada linguagem, visto que, estas oferecem a possibilidade de desenvolver os testes em Ruby, Java ou C++, entre outras linguagens.

Das ferramentas mencionadas, a que seria mais apropriada para ser integrada no processo de desenvolvimento da empresa ABC, seria o *Selenium,* pois permite tirar partido do seu potencial, sem existir a obrigatoriedade de a equipa possuir conhecimentos a nível de programação, enquanto as restantes ferramentas exigem, logo à partida, esse esforço extra que poderia logo condicionar a sua integração. O *SoapUI* poderia ser também, uma opção viável, no entanto, o fato de ser necessário definir os testes antes de estes serem executados, iria tornar-se uma dificuldade para equipa de QA, podendo tornar-se num entrave para a sua utilização. Ao ser feita a integração do *Selenium*, poderia ser criado um plano para fornecer formação à equipa de *Quality Assurance* de forma gradual, com o objetivo de esta conseguir explorar ao máximo todas as possibilidades que seriam presenteadas pelo uso do *Selenium*. Ao ser integrada, esta ferramenta permitiria logo, cobrir a plataforma de *web browsers*, assim como fazer logo um despiste rápido às versões do *software* direcionados para *Smart TV's,* utilizando a versão de *web browser* que é utilizada pelos *developers* para realizarem o desenvolvimento. Com algum tempo investido, acredito, que seria possível integrar a 100% o *Selenium* diretamente com as *Smart TV's*, e que, o resultado final seria um processo de *Quality Assurance* mais rápido e eficiente.

2.5. O ESTÁGIO

2.5.1. Etapas de Trabalho - Métodos

Passarei agora a abordar de forma detalhada, todo o processo de *Quality Assurance* existente na empresa ABC.

Para efetuar a verificação, é necessário, aceder de forma sistemática, a cada Loja Online das várias marcas de dispositivos para os quais a empresa desenvolve aplicações. (Loja da Samsung, Loja da LG, loja da ROKU, etc…), com um conjunto de dados (utilizador e palavra-chave) específicos para cada aplicação. De forma a obtermos informação sobre quais as lojas que se torna necessário aceder para cada aplicação, a equipa de QA tinha um documento Excel disponível *online* através do serviço de Office 365, que continha todas as aplicações já submetidas (independentemente do seu estado atual nas diferentes lojas) e que indicava os dispositivos para os quais, a aplicação tinha sido desenvolvida, assim como, os vários dados necessários para efetuar o acesso às várias lojas.

Em cada acesso realizado, é verificado o estado atual da aplicação perante a Loja, sendo esses estados os seguintes:

- *In* QA ou *Testing* – a aplicação encontra-se em fase de testes por parte da equipa de QA da Loja em questão. Esta ronda de testes é realizada para comprovar que a aplicação cumpre todos os requisites que a Loja obriga para que esta possa ser lançada ao público.

Para além de ser verificado o cumprimento dos requisitos, é também verificada a existência de *bugs* na aplicação que possam comprometer o funcionamento ou qualidade da mesma, sendo que, caso se confirme a existência de *bugs* não críticos para a aplicação, esta pode ser na mesma aprovada, e os *bugs* são reportados para que sejam corrigidos e seja feita uma atualização da aplicação.

- *Waiting for Launch* – a aplicação passou a ronda de testes, e encontra-se à espera que seja iniciado o processo de lançamento ao público.
- *Live* ou *Approved* – a aplicação já se encontra disponivel ao público

- *Rejected, Fail* ou *Resubmission Needed* – foi detetada alguma falha na aplicação que impede que esta seja lançada ao público, o que requer investigação por parte da equipa de QA para determinar o que se passou e qual a acção necessária a tomar para corrigir a situação.

Toda a informação referente às aplicações, lojas e o seu estado, é depois transferida para um documento *word*, que posteriormente é enviado por email, para vários colaboradores (líder de equipa de QA e *project managers*) para que estes estejam atualizados em relação ao estado das aplicações.

Dos vários estados referidos anteriormente, apenas um deles requer algum tipo de ação quando uma aplicação chega a essa situação, sendo esse estado *Rejected, Fail* ou *Resubmission Needed*. Quando este estado é detetado, significada que a aplicação foi rejeitada pela loja, tornando-se necessário determinar qual ou quais os motivos para que isto tenha acontecido.

Os motivos que podem levar à rejeição de uma aplicação por parte de uma das lojas, podem ser o não cumprimento dos requisitos que são obrigatórios para obter aprovação, a descoberta por parte da equipa de *quality assurance* da loja de *bugs* considerados críticos, ou, simplesmente uma discrepância entre as funcionalidades da aplicação e a documentação criada para a mesma. Para o primeiro motivo, a solução passa por verificar qual o requisito ou requisitos que não estão a ser cumpridos, reportar um *bug* na plataforma, notificar o *developer* da aplicação sobre a ocorrência (para que este possa dar alguma prioridade à situação) e, assim que tudo esteja nos conformes, a equipa de *quality assurance* pode avançar com uma nova submissão da aplicação para a loja. Se o motivo da rejeição da aplicação for a descoberta de *bugs*, torna-se necessário a verificação dos mesmos, para que seja possível comprovar se estes realmente ocorrem ou não. Durante a verificação, cada *bug* que é confirmado é reportado na plataforma de *bugs* e, mais uma vez, o *developer* é notificado sobre a situação para tentar dar prioridade à correção destes *bugs*. Após a correção, é iniciado mais uma vez, o processo de submissão da aplicação para a loja. Para o último motivo, o método de correção torna-se diferente, sendo apenas necessário atualizar a

documentação de acordo com as funcionalidades da aplicação e realizar uma nova submissão a mesma.

Na existência de aplicações para serem submetidas para as diversas lojas, torna-se necessário criar vários ficheiros, que seguem as instruções de cada loja, para que estes sejam enviados juntamente com a aplicação.

Após esta primeira tarefa, inicia-se as tarefas internas de testes às várias aplicações. As tarefas são alteradas diariamente, por parte da *team leader de quality assurance* através da atualização semanal de um ficheiro *excel*, que contempla quais as aplicações a ser testadas, em quais plataformas, em que dia da semana e qual o membro da equipa que irá realizar essa tarefa. Estas tarefas podem sofrer alterações ao longo da semana, caso surja tarefas prioritárias.

De uma forma geral, todas estas tarefas eram tratadas como sendo tarefas secundárias, sendo que, estas tarefas seriam realizadas na ausência de *drops* para testar. As *drops* das várias aplicações, são distribuídas pela equipa utilizando o mesmo método das tarefas internas, anteriormente referido.

Passando agora para os procedimentos de testes, irei abordar primeiro os testes de *drops*, e seguidamente, os testes diários realizados.

Na existência de *drops* disponíveis para serem testadas, não existe nenhum procedimento definido sobre como testar ou que tipos de testes realizar de forma a garantir um bom processo de *quality asssurance* em cada *drop* testada, pelo que, o procedimento que irá ser descrito, trata-se do procedimento realizado por mim próprio, e que, considerei ao longo do meu período na empresa ABC, ser o mais adequado e completo. Mas antes de avançar, é necessário definir, o que é afinal uma *drop?* Uma *drop* é simplesmente uma nova versão de uma aplicação que está a ser desenvolvida e que necessita ser testada antes de ser enviada ao cliente ou ser lançada ao público. Avançamos agora para o procedimento de testes:

I. Antes de iniciar qualquer tipo de teste, o primeiro passo, era sempre o de verificar quais as plataformas ou dispositivos para os quais a aplicação estava a ser desenvolvida, através da consulta de informação armazenada em ficheiros disponíveis online, ou, na falta destes, abordar o *project manager* da aplicação e obter essa informação. Isto prevenia a realização de testes em plataformas ou dispositivos desnecessários, evitava a perda de tempo e permitia auferir quais os dispositivos nos quais seria prioritário realizar os testes, isto porque, por vezes o cliente apenas exigia a versão de alguns dispositivos, ao invés de todas as versões desenvolvidas.

II. Após a verificação realizada anteriormente, o passo seguinte era de obter informação sobre a *drop*, visto que, em cada *drop* realizada pelos *developers*, era colocado uma nova mensagem na plataforma online *Basecamp*, que, por sua vez, enviava um e-mail para todas as pessoas associadas ao projeto, contendo a seguinte informação: qual a aplicação, qual a versão, novos componentes, *bugs* resolvidos e notas. Esta informação disponibilizada, permitia-nos verificar qual a versão que estava já disponível nos dispositivos para testar e, se necessário, atualizar para a versão correta, quais os novos componentes que teriam que ser testados, quais os *bugs* que tinham sido resolvidos e, algumas notas referentes a, por exemplo, algum componente não estar concluído a 100% ou a existência de alguma *bug* que os *developers* já tomaram nota, mas que ainda não conseguiram resolver.

III. Já com a versão correta nos diferentes dispositivos, e com a informação necessária toda obtida, podemos então iniciar os testes propriamente ditos. De referir que todas as plataformas e *software* de apoio referidas durante esta fase, serão abordadas posteriormente neste relatório com mais detalhe.

IV. Primeiramente, eram verificados os *bugs* incluídos na *drop*. Para tal, era feito o acesso à plataforma online de controlo de *bugs,* Mantis, onde estava toda a informação referente aos *bugs*, desde qual era o *bug* em si, quais os passos a

seguir para se conseguir reproduzir esse *bug*, qual a solução atual do *bug* (aberto, resolvido, reaberto, impossível de reproduzir, não resolúvel, duplicado, não são necessárias alterações, suspenso ou não será resolvido), tipo de *bug* (funcionalidade, trivial, texto, ajuste, pequena, grande, *crash* ou bloqueio), o seu estado (novo, resposta, reconhecido, confirmado, atribuído, resolvido ou encerrado), e qual a prioridade (nenhuma, baixa, normal, alta, urgente ou imediato). Estes também podiam conter anexos como fotos ou vídeos, de modo a auxiliar que qualquer pessoa conseguisse rapidamente perceber, em concreto, de que se tratava aquele *bug*.

O estado atual do *bug*, normalmente definia qual a ordem para se rever os *bugs*, sendo que, teria inicio pelos *bugs* com solução atual de resolvido. Para verificar estes *bugs*, era simplesmente tentar replicar o *bug*, seguindo os passos que estavam descritos no ticket, e se não fosse possível replicar o bug, atualizar o seu estado para encerrado automaticamente fechava o bug no Mantis.

Em situação inversa, na qual ainda era possível replicar o bug apesar da sua solução estar marcada como resolvido, esta era atualizada para reaberto. Este processo repete-se para todos os bugs com solução de resolvido. Para a solução de impossível de reproduzir, o processo referido no paragrafo anterior era repetido também para esta solução.

Para as soluções de não resolúvel, não será resolvido, suspenso e não são necessárias alterações, era necessário obter informação junto do *developer* e *project manager*, de forma a ser possível colocar uma nota para justificar, o porquê de aquele bug estar com aquela determinada solução.

V. Terminada a verificação dos bugs associados à *drop*, passávamos então, à verificação dos novos componentes desenvolvidos. Para isto, recorríamos ao *workflow* da aplicação, para sabermos qual o comportamento dos componentes, e também, recorríamos aos *mockup's* da aplicação, de forma a validar todo o

design implementado. Ao longo de todo este processo, todos os bugs detetados vão sendo apontados num ficheiro, assim como, são tiradas fotos e gravados vídeos que posteriormente serão anexados aos respetivos bugs.

VI. É realizado um teste geral a toda a aplicação, de maneira a garantir que os novos componentes não deram origem a novos bugs ao serem integrados com os componentes desenvolvidos anteriormente

VII. Para terminar, os bugs apontados são reportados na plataforma Mantis, com toda a devida informação acrescentada a cada um.

Como procedimento final a nível interno, após terminado todo o processo que foi acabado de ser descrito, era colocada uma nova mensagem no Basecamp, contendo qual o projeto que foi verificado, quais os bugs que foram fechados durante todo o processo, quanto e quais foram os bugs que foram reportados e reabertos, quais os componentes que foram testados e se estes estavam de acordo com o esperado.

Para finalizar todo este processo, toda a informação disponibilizada no Basecamp pela equipa de *quality assurance*, era depois enviada ao cliente do projeto, juntamente com a respetiva versão da aplicação, ficando depois, a cargo do *project manager*, reportar qualquer feedback realizado por parte do cliente.

2.5.2. Descrição Crítica do Processo em Etapas

Ao longo da realização deste relatório, surgiu a oportunidade de abraçar um novo desafio numa empresa de maior dimensão (será designada por empresa DEF ao longo do capitulo), pelo que, resolvi aceitar esta oportunidade. Ao longo deste capitulo, irei expor a minha critica pessoal a todo o processo mencionado no capitulo anterior, e para finalizar, irei expor como é aplicado o processo de *quality assurance* na nova empresa que me encontro atualmente, de forma a que seja possível existir um termo de comparação entre duas realidades completamente distintas.

Todo o processo descrito no capitulo anterior, está longe de ser perfeito, tratando-se de um processo que simplesmente satisfaz a necessidade da empresa no imediato e não contempla necessidades futuras de melhoramento, existindo apenas essa preocupação quando a situação assim o exige.

Irei começar por abordar a falta de organização e falta de informação relativamente à documentação de projetos. Para todos os projetos, existem os vários *mockup's* que definem todo o design da aplicação, assim como um *workflow* que explica qual o comportamento desejado pelo cliente para todos os componentes da aplicação, e, sendo esta uma informação essencial para todo o desenvolvimento, deveria estar centralizada num único local de forma a facilitar o seu acesso quando esta é necessária. No entanto, tal situação não se verifica, sendo os *mockup's* armazenados num local diferente do *workflow*, e por vezes, existir várias versões de *mockup's* e *workflow* armazenados em vários locais, sem que seja possível determinar qual a versão correta a utilizar. Relativamente à falta de informação, isto refere-se, ao fato de existirem alterações constantes aos *mockup's* e *workflow*, sem que estas sejam comunicadas à equipa, provocando atrasos no desenvolvimento, devido aos componentes serem desenvolvidos sem estarem a cumprir os requisitos necessários ou o design pretendido.

Passando agora para a equipa de *quality assurance*, iniciarei a minha critica à forma como as tarefas eram organizadas. As várias tarefas a serem realizadas ao longo de cada semana, eram descriminadas num ficheiro *excel* e disponibilizadas online. Esta tarefas apenas indicando que uma aplicação deveria ser testada, mas não informada que tipo de teste realizar ou quais as plataformas a testar, sendo sempre necessário haver constante procura de informação por parte dos membros da equipa.

Referente ao método de testar as várias aplicações, existem bastantes falhas a ser apontadas, começando pelo fato de não existir nenhum método definido nem existir *guide lines* para tal. Todos os testes realizados, eram basicamente testes de integração e testes de regressão, todos baseados na experiencia em testes anteriores dos vários membros da equipa.

É importante referir a falta de uso de *test cases* e de tipos de testes diferentes, não sendo o caso de não existirem ou não existir o conhecimento dos mesmos, mas sim, a simples falta de incentivo para tal, na qual a equipa era incentivada durante uma semana a usar *test* cases e testes diferentes, como depois, a equipa estava duas ou três semanas sem fazer uso destas mais valias. De referir que, os *test cases* raramente eram atualizados, pois os que existiam eram referentes a projetos bastantes antigos.

Para além da falta de incentivo para usar *test cases* e diferentes tipos de testes, também o uso de ferramentas que pudesse auxiliar todo o processo de testes também não era incentivado por parte da empresa.

Tudo isto que acabou de ser referido, apenas contribuía para que os testes realizados não fossem testes de qualidade que transparecessem 100% confiança nos seus resultados.

Outra falha relativa ao método de testar as várias aplicações, era a quantidade de testes realizados a uma aplicação, chegando, por vezes, a serem realizados testes a uma aplicação todos os dias durante uma semana de trabalho, sem que existisse justificação para tal, dando a entender à equipa que se tratava simplesmente de tarefas para nos manter ocupados devidos à falta de *drops* para testar. Como consequência, a motivação e qualidade dos testes realizados pela equipa de *quality assurance* era fortemente afetada, pois todo o desempenho da equipa era avaliado de acordo com a quantidade de bugs encontrados, o que, de certa forma, levava a equipa a um pequeno nível de desespero, tentando encontrar mais um bug para poder justificar aquele tempo perdido a testar uma aplicação já testada mil e uma vezes.

Adicionando a tudo o que já foi referido, juntámos a falta de formação da equipa na área de testes, a falta de incentivo por parte da empresa para corrigir tal situação. Os membros da equipa não tinham formação adequada para desempenhar as funções pretendidas, e, sempre que um novo membro era adicionado à equipa, este era formada com base na experiencia de cada membro da equipa, o que não garantia uma boa formação.

O aspeto mais negativo, é o fato de toda a equipa de *quality assurance* ser tratada como um departamento à parte das restantes equipas, que apenas é necessário na fase final de cada ciclo de desenvolvimento. Em capítulos anteriores deste relatório, já foi referido e demonstrado as várias vantagens de se aplicar *quality assurance* ao longo de todo o processo de desenvolvimento, ao invés de o guardar para o fim, mas infelizmente a mentalidade da empresa, apesar de estar apontada na direção correta, ainda não se move nessa direção, tentando transparecer um ambiente de desenvolvimento totalmente *agile*, quando na realidade, acaba por ser mais um ambiente de desenvolvimento em cascata com algumas vertentes de *agile*.

Vejamos agora, a outra face da moeda, numa empresa que utiliza o processo de *quality assurance* de uma forma mais correta quando o desenvolvimento é realizado em ambientes *agile*.

Como método de desenvolvimento *agile*, a empresa DEF utiliza SCRUM, um método de desenvolvimento iterativo e incremental, baseado em *sprints (*uma *sprint* é um intervalo de tempo, no qual, uma equipa tem o objetivo de concluir o desenvolvimento de alguns componentes que fazem parte de um projeto*)* de desenvolvimento dos seus projetos, em que, estes são divididos em partes mais pequenas, de forma a ser possível conclui-los ao longo de várias *sprints* (de momento, as *sprints* estão estabelecidas como períodos de duas semanas).

Antes de descrever este ambiente de desenvolvimento, torna-se necessário referir a constituição de cada equipa, sendo que, cada equipa é composta por dois *developers* de aplicações, dois *developers* de API, dois *quality assurance* (um manual, direcionado para o *front end* do componente e um de automação, direcionado para o *back end* do componente) e um *project manager*. Cada equipa trabalha com um *product owner* diferente, de acordo com o projeto que é atribuído.

Ao iniciarmos um novo ciclo de desenvolvimento, o *project manager* executa o levantamento de requisitos em conjunto com o *product owner*, dividindo o projeto em componentes, definindo os requisitos necessários para aprovação dos mesmos e estabelecendo o comportamento de cada componente.

Findo o levantamento de requisitos, o *project manager* transforma os requisitos em *user stories* (é possível dois exemplos de *user stories* na figura 10). Isto não é mais do que transformar os requisitos e o comportamento de cada componente numa só fonte de informação, em que, é descrito de forma mais simples e concisa, quais as ações que um utilizador deve poder realizar em cada componente.

Após esta transformação de informação, é marcada uma primeira reunião, designada de *planning*, na qual, com a presença de toda a equipa, são definidas as *user stories* que irão fazer parte da próxima *sprint*.

Figura 10 - Nesta imagem temos dois simples exemplos de uma *user story*. Uma *User story*, é uma breve explicação de um meio para atingir um objetivo, sem grandes detalhes de como esse meio deve ser ou como vai ser utilizado.

Uma segunda reunião é marcada para o dia seguinte, designada de *grooming*. Nesta reunião, as *user stories* que foram definidas para a *sprint,* são analisadas para verificar que toda a informação necessária se encontra introduzida em cada *user story*, e que esta está clara (nesta fase, *quality assurance* pode já identificar requisitos em falta, faltas de informação e prever cenários que podem originar falhas). Para além disto, as *user stories* também são analisadas para se determinar se é possível serem divididas em *user stories* mais pequenas, permitindo que estas sejam realizadas ao longo de várias *sprints* de forma eficaz.

Iniciando agora o desenvolvimento, o QA manual começa a escrever as suas *testing notes*, para quando realizar os testes, ter uma forma de validar que o *front end* do componente está desenvolvido de acordo com o pretendido. Ao mesmo tempo, o *QA* de automação começa a preparar também os cenários de teste necessários, para este também poder validar o *back end* do componente. Os cenários e *testing notes*, ao serem iniciados ao mesmo tempo que o desenvolvimento do componente por parte dos *developers*, permite um novo refinamento dos requisitos e do comportamento, podendo identificar novas falhas, permitindo que sejam logo corrigidos nas *user stories*, e prevenindo que os *developers* tenham atrasos de grandes dimensões durante o desenvolvimento.

Terminado o desenvolvimento, toda a fase de testes é feita em três partes:

I. Uma primeira fase de testes é realizada num ambiente de testes, que simula a 100%, o ambiente final no qual o componente ou componentes serão lançados ao público. Como em qualquer fase de testes, é verificado que o componente cumpre os requisitos estabelecidos, apresenta o comportamento esperado, e, é validado pelas *testing notes* e cenários criados pelos *QA's*. Qualquer *bug* identificado, é discutido com o *project manager* e com o *product owner* para decidir se, este é um fator que impeça o componente de ser aprovado pelos *QA's*, e, caso isso se verifique, o *bug* é reportado e o componente é reprovado. Em situação positiva, o *bug* é reportado para ser resolvido numa *sprint* futura, e o componente é aprovado.

II. Na segunda fase de testes, os componentes aprovados na fase anterior, são agora testados num ambiente de *stagging*, que é basicamente uma cópia do ambiente final, mas que utiliza em tempo real os dados e informação do ambiente final. Todo o processo de testes referido anteriormente, é repetido e o procedimento a seguir também é o mesmo.

III. Numa terceira e final fase de testes, os componentes são testados no ambiente final, utilizado pelo público. Repete-se todos os passos descritos nas fases anteriores, assim como os procedimentos. A única exceção nesta fase, é referente à situação de aprovação de um componente, pois este ao invés de avançar para uma nova fase de testes ao ser aprovado, é simplesmente fechada a *user story* referente a esse componente, sendo que, qualquer *bug* que tenha sido identificado e que seja não esteja relacionado com este componente, será resolvido numa *sprint* futura, como forma de melhoramento e manutenção do componente. Caso seja detetado algum *bug* que, de alguma forma, afeta o componente em questão, este é reportado na plataforma devida, e a *user story* é reaberta, para que seja corrigido(s) o(s) *bug*(s) encontrados.

Terminada a *sprint*, é realizada uma reunião, onde são discutidos os aspetos positivos e negativos referentes à *sprint*, assim como um balanço do desempenho da equipa. São também definidas ações de ataque aos aspetos negativos, de forma a que estes vão sendo corrigidos ao longo das *sprints*.

2.5.3. Ferramentas de Apoio

Existem diversas ferramentas e processos internos que, no dia-a-dia, ajudam um *quality assurance* a poder desempenhar o seu papel e dar o seu contributo no desenvolvimento de software. Iremos agora conhecer estas ferramentas e processos de uma forma mais detalhada e, descobrir como são utilizadas numa constância diária.

2.5.3.1. MANTIS BUG TRACKER

O *Mantis Bug Tracker* é uma ferramenta online utilizada para realizar a gestão dos bugs de uma aplicação.

Trata-se de uma aplicação baseada na WEB, que cria versões especificas de acordo com o pretendido pelos clientes que compram os seus serviços.

No âmbito da empresa ABC temos uma versão do *Mantis bug Tracker* criada especificamente para os nossos projetos, na qual temos total controlo sobre a plataforma, desde da criação de novos projetos, novos utilizadores, alterar as diferentes opções que podem ser selecionadas e alterar o nível de permissões e cada utilizador.

Esta plataforma é usada principalmente pelos *testers*, é nesta plataforma que são reportados todos os bugs encontrados quando é testada uma nova versão de um software que está a ser desenvolvido. (é possível ver um exemplo da interface da plataforma Mantis na figura 11). [22]

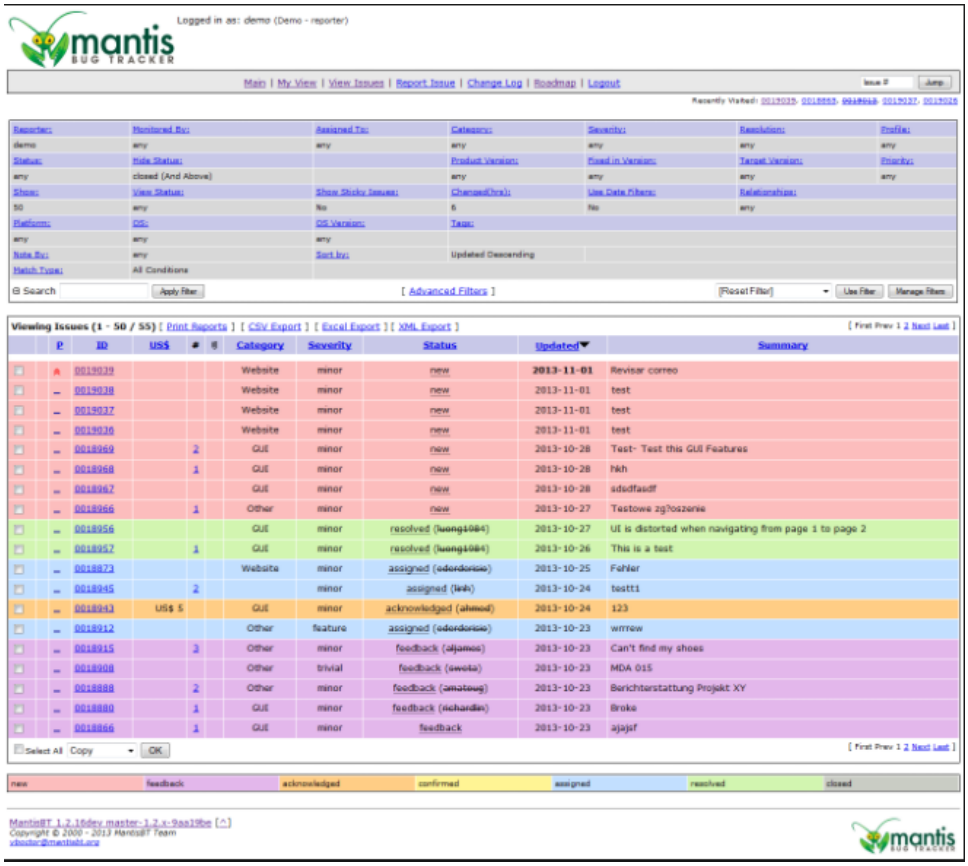


Figura 11 - Exemplo da *interface* da plataforma de *bug tracking* Mantis, utilizada pela empresa ABC.

2.5.3.2. JIRA

O JIRA é outra plataforma online para fazer *bug tracking*.

Assim como o *Mantis Bug Tracker*, trata-se também de uma ferramenta paga, que cria uma versão especifica para cada cliente, onde este faz toda a gestão dos bugs relacionados com os seus projetos.

O JIRA destaca-se do Mantis por também incluir ferramentas de gestão de projetos, como incluir tarefas, definir prazos para as mesmas e ser possível ver o estado atual de cada tarefa (iniciada, não iniciada, interrompida, entre outros).

O JIRA também sobressai no nível de detalhe no que toca a *bug tracking*, mostrando aos utilizadores um historio completo ordenado desde da data de introdução do bug na plataforma, até a atualidade, todos os estados que o bug já passou, todos os comentários associados a este e permite também selecionar um estado mais detalhado para o bug. [23]

2.5.3.3. Plataforma XYZ

A plataforma XYZ é o "motor" da empresa ABC, tratando-se de uma plataforma criada por um dos fundadores da ABC em conjunto com outros colegas e que permite o rápido desenvolvimento de aplicações para várias plataformas ao mesmo tempo, nomeadamente *Smart TV's*.

É através desta plataforma que são geradas a maior parte das versões de todos os projetos que são desenvolvidos pela ABC, apenas plataformas WEB, *Android* e IOS não são geradas através desta plataforma.

O funcionamento é bastante simples, o *developer* envia o código que desenvolveu para a plataforma, seleciona quais os dispositivos que deseja que o código seja gerado e a plataforma trata do resto. Em poucos minutos temos o código pronto para ser utilizado em várias *smart* TV's diferentes (LG, Panasonic, Samsung, Philips, Sharp, Amazon *Fire* TV, Android TV, Sony, OperaTV).

Cada plataforma tem um link especifico criado para poder ser acedido via WEB através de uma aplicação criada internamente para este propósito.

2.5.3.4. BASECAMP

O BASECAMP, é uma ferramenta online para gestão de projetos, comunicação entre equipas de desenvolvimento e comunicação com os clientes.

Esta ferramenta é utilizada na ABC para realizar as várias *drops* durante o ciclo de desenvolvimento dos projetos.

Também serviu inicialmente para centralizar toda a informação relativa aos projetos desenvolvidos, desde *mockup's* a *workflow*.

Grande parte do feedback por parte do cliente também é feita através desta ferramenta. [24]

2.5.3.5. Filezilla

Como a plataforma XYZ utiliza ficheiros JSON alojados num servidor, é necessário editar estes ficheiros cada vez que queremos lançar uma aplicação em alguma *Smart TV*.

Para tal, utilizamos o Filezilla, que é um cliente de *FILE TRANSFER PROTOCOL* (FTP) gratuito, *cross-platform* e bastante simples de utilizar, necessitando apenas de ser configuração a ligação ao servidor que desejamos, e após esta ligação ser feita com sucesso, só temos que encontrar o ficheiro que queremos editar.

Ao terminar a edição, o Filezilla pergunta ao utilizador que queremos substituir o ficheiro existente no servidor pelo que acabamos de editar, e, caso selecionemos que sim, este faz o *upload* e a substituição automaticamente, ficando o ficheiro atualizado e pronto a ser acedido pelo *ABC LAUNCHER*. (é possível ver um exemplo de interface da ferramenta FileZilla na imagem 12). [25]

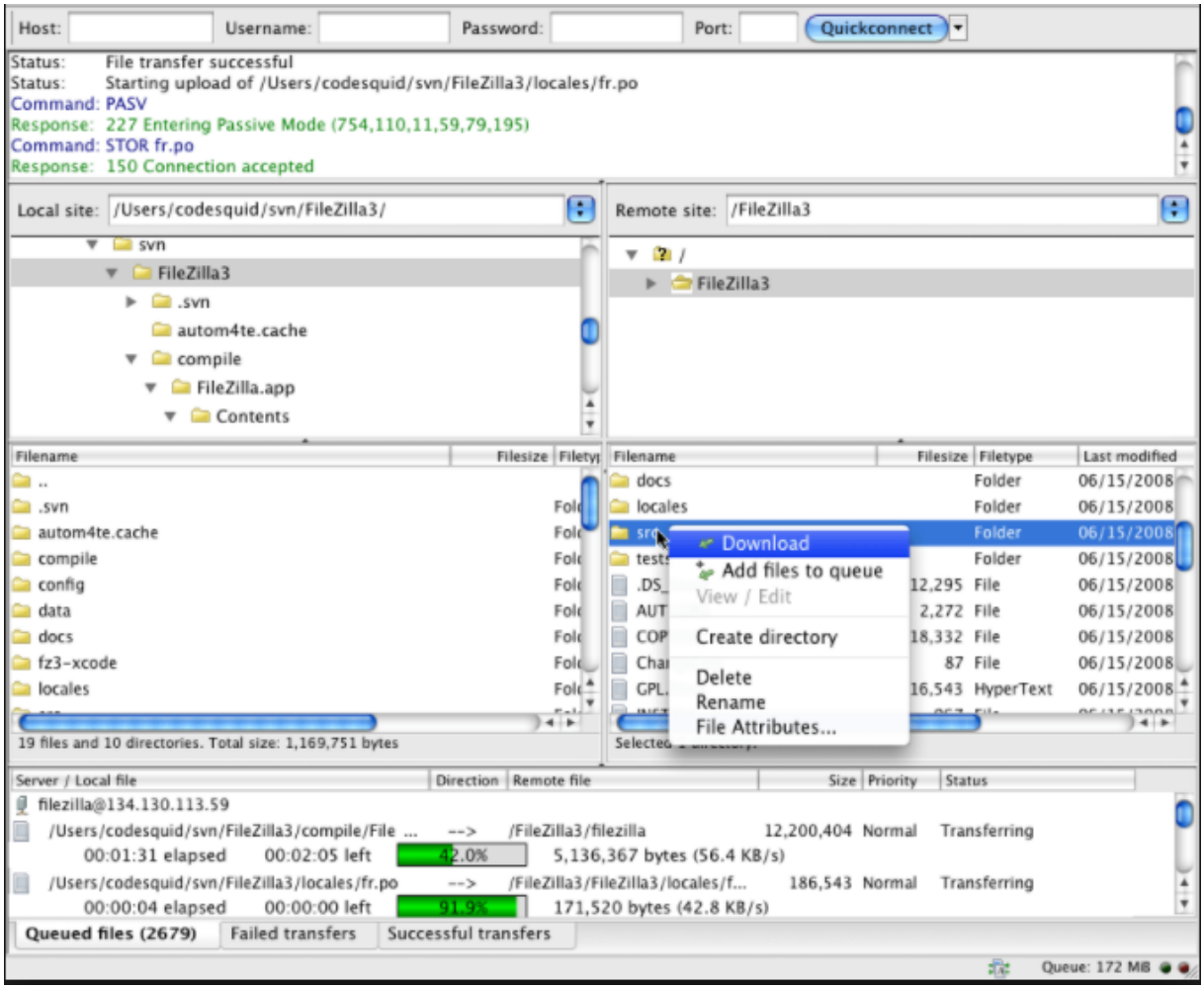


Figura 12 - Exemplo da interface da plataforma FileZilla, utilizada para envio de versões das aplicações para os vários dispositivos existentes na empresa ABC.

2.5.3.6. BETA by CrashLitycs

BETA by Crashlytics é uma aplicação 2 em 1 perfeita para desenvolvimento de software a nível mobile (IOS e Android), sendo já considerada um kit de desenvolvimento de software (SDK) e um dos mais usados a nível mundial.

Numa primeira fase, o objetivo deste SDK é facilitar o teste de aplicações na fase BETA ou 2ª fase do ciclo de desenvolvimento de software. Nesta fase, as aplicações são testadas por utilizadores para verificarem a integridade, estabilidade e usabilidade das aplicações desenvolvidas. Isto torna-se bastante simples através do BETA, pois trata-se apenas de uma aplicação instalada no dispositivo do utilizador, e através desta, o

utilizador recebe consegue fazer download e instalar a aplicação que quer testar diretamente no seu dispositivo.

Numa segunda fase, o Crashlytics torna-se o melhor amigo do *developer*, pois a cada erro ou crash que ocorra na aplicação, esta aplicação envia um relatório para o *developer* a indicar o tipo de erro que ocorreu e qual a linha de código onde este erro ocorreu.

O processo completo ocorre da seguinte maneira:

I. O *developer* regista a sua aplicação no Crashlytics e faz *upload* da versão que quer que seja testada (Android ou IOS).

II. Numa interface própria do Crashlytics, o *developer* coloca o email dos *testers* e envia um convite para dar acesso a estes para fazerem download da aplicação (este convite só é necessário ser enviado 1 vez)

III. Os *testers* recebem um email com o respetivo convite. Ao clicarem, este abre automaticamente uma pagina web, na qual terão que fazer download da aplicação BETA (a figura 13 mostra um exemplo de um destes convites).

IV. Depois de instalada, o BETA mostra a aplicação para a qual foram dadas permissões com a informação que existe uma versão disponível para download. Ao fazer o download, a aplicação é instalada no dispositivo para testes, e o *tester* tem acesso há informação relativa ao que pode testar naquela versão, o que já foi desenvolvido e o que ainda está por desenvolver.

V. Sempre que o *developer* fizer *upload* de uma nova versão, os *testers* recebem uma notificação (caso tenham a aplicação instalada no seu dispositivo) e email a informar que está disponível uma nova versão para download.

VI. Durante os testes, sempre que ocorrerem erros, os *developers* recebem os relatórios com os erros, como foi mencionado já anteriormente (exemplo desta

interface na figura 14).

Isto permite fazer rapidamente *deploy* das versões para testar e ajuda os *developers* a rapidamente corrigiram a maior parte dos erros detetados, reduzindo o tempo de desenvolvimento das aplicações. Também facilita os testes, pois trata-se de método bastante simples de os *testers* obterem as versões para testarem assim como a informação necessária para um teste bem-sucedido. [26]

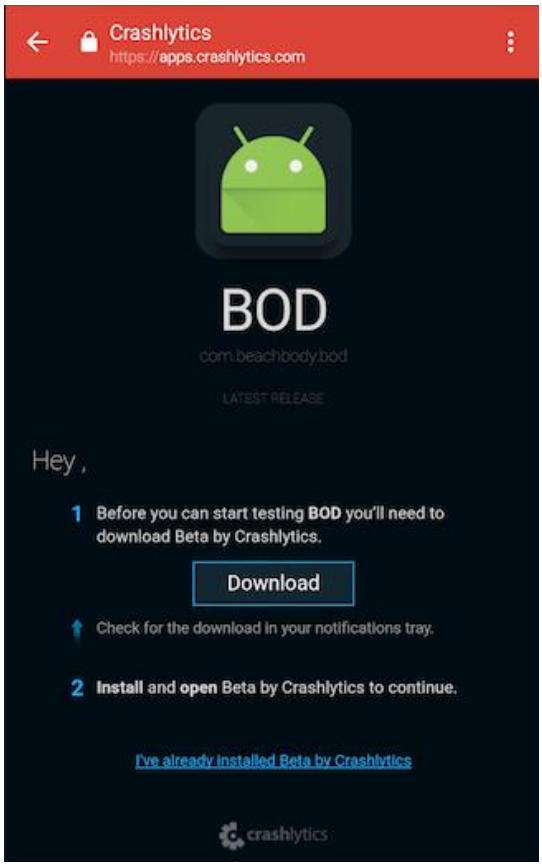


Figura 13 - Nesta primeira imagem, temos um exemplo de um email recebido a convidar o utilizador para instalar uma determinada aplicação

Figura 14 - Nesta segunda imagem, temos um exemplo da interface que esta ferramenta disponibiliza ao *developer*, permitindo controlar, por exemplo, número de utilizadores ativos por dia, número de utilizadores novos por dia, entre outros dados.

2.5.3. TestCases

Um TEST CASE, é um conjunto de ações pré-definidas, condições e resultados esperados, previamente preparados para atingir um determinado objetivo, como por exemplo, verificar se uma determinada ação tem o resultado correto no software.

Este são avaliados de forma simples com PASS e FAIL se o resultado obtido no software é o esperado ou não.

Utilizámos os TEST CASES principalmente para testar ações que são comuns à grande parte de software que desenvolvemos, e também são utilizados para fazer uma verificação mais detalhada antes de fazermos alguma submissão para as *stores*. (é possível ver um exemplo de test cases na figura 15)

Summary	Preconditions	Test Steps and Description	Expected Results	Test Results	Notes	bug ID
Sign In from homepage	Application is installed.	1.Launch application and open homepage. 2.Press 'Sign In' button in homepage.	2.Load and display Sign In page.	Fail		1
Sign In from adv page	Application is installed.	1.Launch application and open adv.page. 2.Press 'Sign In' button.	2.Load and display Sign In page.	Pass		
Sign In from slide menu	Application is installed.	1.Launch application and open slide menu. 2.Select 'Sign In'in the menu list.	2.Load and display Sign In page.	Pass		
Sign In from register page	Application is installed.	1.Launch application and open slide menu. 2.Select "Register" from menu list. 3.Press 'Sign In' button in register page.	3.Load and display Sign In page.	Pass		
User account check in Sign In	Application is installed.	1.Launch application and open homepage. 2.Press 'Sign In' button in homepage. 3.Check user account interface.	3.The user account input displays with default text "Mobile Number or Email Address"	Pass		
Password check in Sign In interface	Application is installed.	1.Launch application and open homepage. 2.Press 'Sign In' button in homepage. 3.Check password input interface.	3.The password input display with default text "Password".	Pass		
Forgot Password in Sign In interface	Application is installed.	1.Launch application and open homepage. 2.Press 'Sign In' button in homepage. 3.Check 'Forgot Password' input interface.	3.Default click links to open Forgot Password page.	Pass		
Register in Sign In interface	Application is installed.	1.Launch application and open homepage. 2.Press 'Sign In' button in homepage. 3.Check password input interface.	3.Default click links to open Register page. And texts 'Need an account?' display above the link texts 'Register'	Pass		
Slide menu in Sign In	Application is installed.	1.Launch application and open homepage.		Pass		

Figura 15 – exemplo de test cases que são utilizados para realizar testes ao software e garantir que estes cumprem os requisitos especificados.

2.5.3.8. POSTMAN

Esta ferramenta é utilizada no teste de *API's*. Como a empresa ABC começou a produzir os seus próprios CMS para cada projeto que desenvolve, de forma a reduzir o tempo de desenvolvimento do software, tornou-se necessário começar a realizar testes na API.

O POSTMAN foi criado com o propósito de ajudar no desenvolvimento de *API's* e também para a realização de testes. Após os *developers* fornecerem os pedidos para colocarmos no POSTMAN, conseguimos ver o que é devolvido do CMS em cada pedido e comparar com o conteúdo que as aplicações estão a mostrar, o que nos permite determinar, em caso de erro, se é da aplicação, do CMS ou até de ambos,

permitindo a nós, *QA's*, fazer uma deteção bastante exata da origem do erro. (é possível ver um exemplo de interface e de utilização da ferramenta Postman na figura 16). [27]

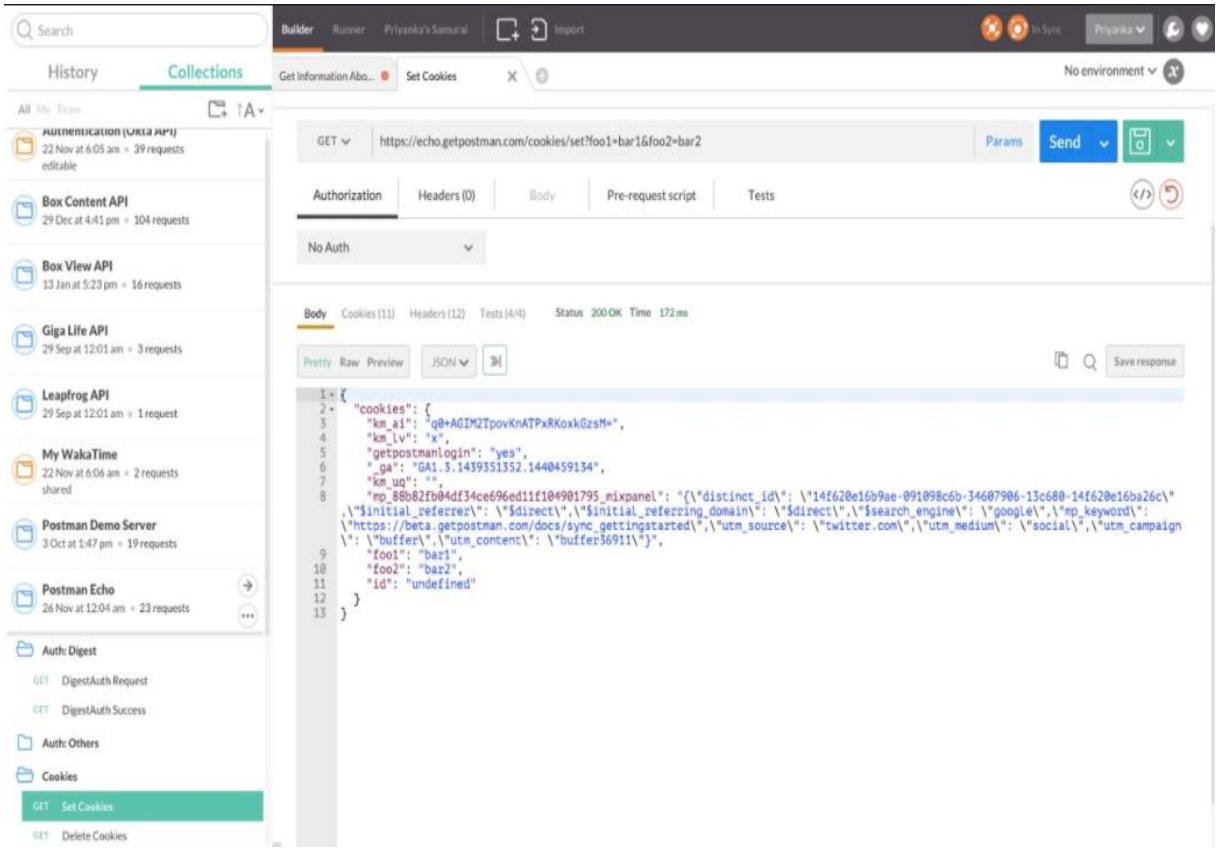


Figura 16 - Um exemplo de uma verificação de API utilizando a ferramenta Postman. Na imagem, podemos quando o pedido efetuado e a respetiva resposta.

2.5.3.9. Workflow

O WORKFLOW é um documento que é produzido em conjunto pelo cliente e pelos designers, no qual está descrito todo o funcionamento pretendido para a aplicação por parte do cliente.

Este documento é importantíssimo para nós *QA's* pois, através deste, conseguimos saber qual o comportamento esperado para a aplicação e na realização de testes identificar erros de comportamento.

Este documento também nunca é definitivo, sofrendo sempre alterações ao longo do desenvolvimento da aplicação, por vezes por vontade do cliente em alterar algum comportamento que não lhe agrada quando implementado, outras vezes de forma a reduzir o numero de passos que o utilizador teria que fazer para realizar determinada ação ou simplesmente para melhorar a usabilidade geral da aplicação. (é possível ver dois exemplos de Workflow's nas figuras 17 e 18).

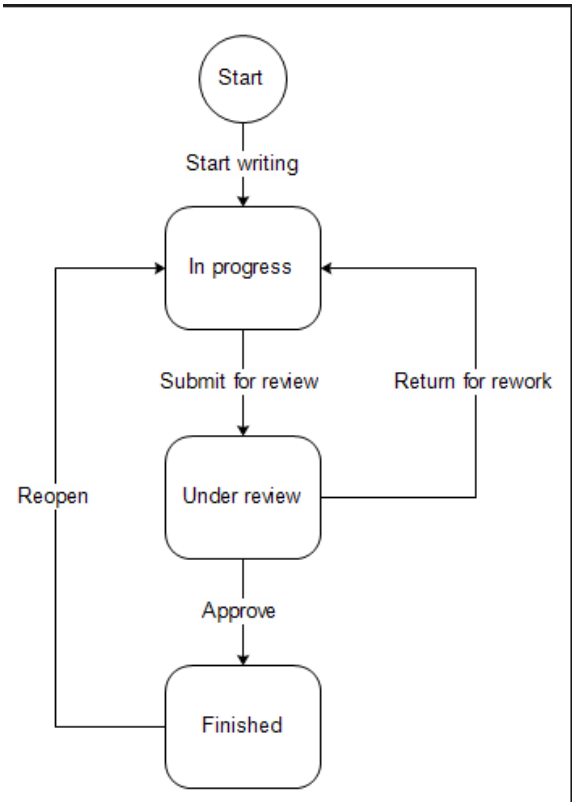
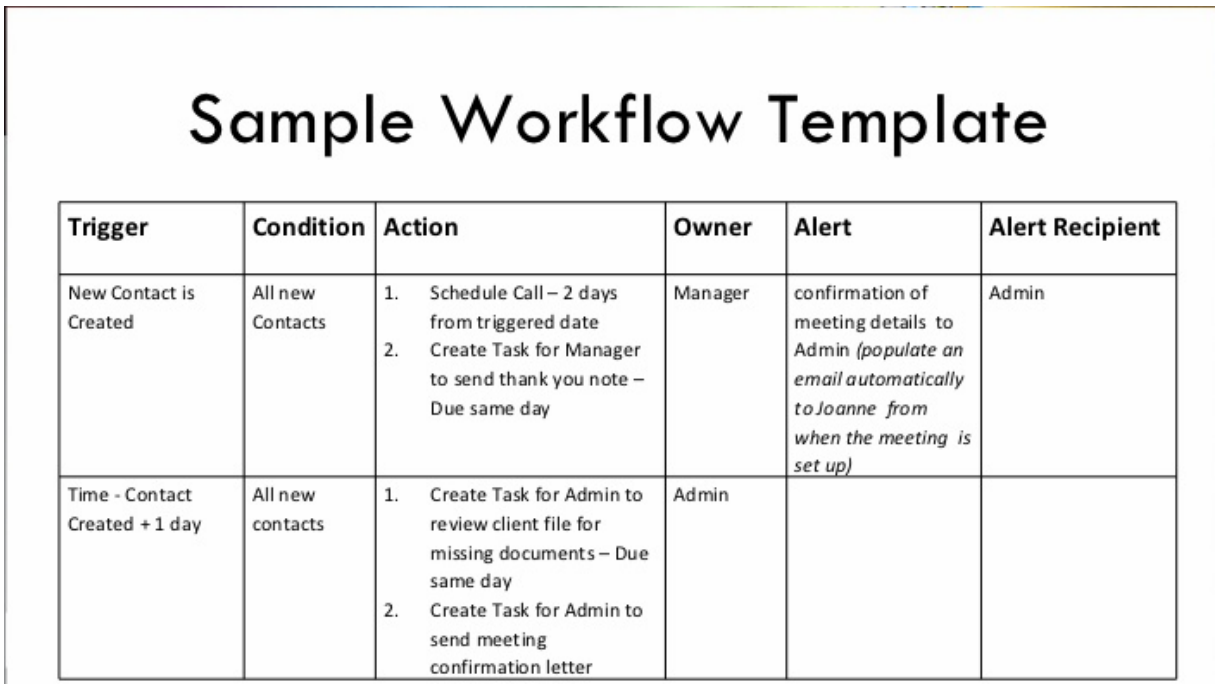


Figura 17 - Um exemplo bastante simples de um *workflow*, referente a um processo de escrita que é submetido a uma revisão.



Sample Workflow Template

Trigger	Condition	Action	Owner	Alert	Alert Recipient
New Contact is Created	All new Contacts	1. Schedule Call – 2 days from triggered date 2. Create Task for Manager to send thank you note – Due same day	Manager	confirmation of meeting details to Admin *(populate an email automatically to Joanne from when the meeting is set up)*	Admin
Time - Contact Created + 1 day	All new contacts	1. Create Task for Admin to review client file for missing documents – Due same day 2. Create Task for Admin to send meeting confirmation letter	Admin		

Figura 18 - Temos agora um exemplo mais complexo de um *workflow*, no qual, temos ações associadas a determinados eventos, qual o funcionário que deve executar cada ação e inclusive, qual o alvo de uma das ações.

2.5.3.10. Mockup's

Os Mockup's são as várias imagens que mostram qual o design final pretendido para a aplicação.

Claro que os Mockup's iniciais nunca correspondem à versão final da aplicação, pois esta sofre sempre alterações de design ao longo do seu desenvolvimento.

2.5.3.11. One Drive / Dropbox / Microsoft Sharepoint

Estas 3 ferramentas de partilha de ficheiros são as usadas dentro do mundo TVAPP.

O *one drive* inicialmente era a aplicação usada para centralizar toda a informações relativa aos projetos desenvolvidos, mas com o passar do tempo, tornou-se necessário encontrar uma ferramenta mais completa, pelo que, optou-se pelo Microsoft Sharepoint. Esta nova ferramenta permite alojar uma grande variedade de ficheiros e também integrar a Dropbox. A forma como o SharePoint está a ser utilizado é a seguinte:

- Ficheiros de texto, *excel*, PDF e informações relativas aos projetos estão alojados diretamente no SharePoint
- Tudo o que for relativo a *mockup's* e imagens, encontra-se no Microsoft Sharepoint uma pasta que ao clicar, abre uma nova pagina na Dropbox para download do que é pretendido

Este tipo de organização permite maximizar o espaço e também rapidez quando necessário aceder a algum tipo de informação relativa aos projetos.

Apesar disto, a equipa de *QA's* continua a utilizar a Dropbox para alojar vídeos relativos a bugs encontrados nas aplicações que requerem um certo numero de passos para serem reproduzidos, de forma a partilharem com os *developers*, para estes conseguirem perceber e corrigir o bug mais rapidamente [28, 29, 30]

2.5.3.12. Skype / Microsoft Teams

Como a TVAPP se encontra espalhada pela Europa, tendo a sua sede na Inglaterra, *developers* em Espanha, Portugal Continental e Ilha da Madeira, a comunicação e coordenação torna-se um aspeto essencial entre as várias equipas e difícil de conseguir se não estiverem disponíveis ferramentas que auxiliem. Para isto, a ABC utiliza a ferramenta de comunicação pela internet SKYPE, uma ferramenta gratuita e a mais popular em todo o mundo.

Através desta, é feita toda a comunicação entre as várias equipas e reuniões com os clientes quando não é possível serem feitas com a presença física da pessoa.

No inicio do ano, a Microsoft lançou a ferramenta TEAMS, uma plataforma de concorrência ao SKYPE e que foi integrada dentro do conjunto de *office productivity suite* ou OFFICE 365 que é um serviço subscrito pela empresa. [31, 32]

2.5.4. Plataformas e Dispositivos de Testes

Devido à diversidade de desenvolvimento efetuado na empresa ABC, no que diz respeito a plataformas e dispositivos, existe a necessidade de, conseguir disponibilizar o maior número de plataformas e dispositivos possíveis à equipa, para que, seja possível incluir estes dispositivos no processo de desenvolvimento de software, nomeadamente na fase de desenvolvimento, com os *developers*, a terem plataformas reais para irem verificando o software desenvolvido, assim como, para a fase de testes, na qual a equipa de *quality assurance* teria acesso a plataforma e dispositivos reais para realização dos seus testes, aproximando, o nível de utilização, o mais possível do nível de utilização final.

Com esta necessidade em mente, o espaço de trabalho encontrava-se repleto de plataformas e dispositivos, tudo para auxilio do processo de desenvolvimento. Em maior número, temos televisões inteligentes, mais conhecidas como *Smart Tv's*, que constituem a principal plataforma para a qual todo o desenvolvido é efetuado. O elevado número de *Smart Tv's* presentes no local de trabalho, deve-se à quantidade de marcas, modelos e sistemas existentes no mercado, e também, pelo fato do software desenvolvido, abranger grande parte destas plataformas.

Para se conseguir ter uma noção da quantidade de *Smart Tv's* existentes no mercado, providencio, o seguinte exemplo: usando a marca LG, que contempla vários modelos de *Smart tv's* diferentes, mas que, para além de estes modelos, também contempla sistemas diferentes para as suas *Smart Tv's*, sendo que, os modelos desde 2012 até 14, utilizando sistema NetCast, enquanto os modelos a partir de 2015, já utilizam sistema WebOS que, atualmente, apresenta como versão mais recente, a versão 3.5. Estes sistemas, podem ser comparados ao sistema operativo de um computador, e, com cada versão que vai sendo lançada no mercado, estas podem apresentar novas funcionalidades, o que, para quem desenvolve software para este tipo de plataformas, obriga a adquirir os modelos mais recentes, para poder explorar

as novas funcionalidades, e, criar software compatível com essas mesmas funcionalidades.

Partindo do exemplo dado, temos agora que considerar, que existem imensos fabricantes de *Smart Tv's*, como Samsung, Lg, Sony, Hisense, Sharp, Panasonic, entre outros, em que, cada marca apresenta os seus próprios sistemas operativos para as suas *Smart Tv's,* com as suas funcionalidades, algumas vezes, especificas da marca, e também, vários modelos diferentes, conseguimos ter uma boa noção da quantidade de plataformas deste género que é necessário ter numa empresa que desenvolve software para as mesmas.

O exemplo apresentado para as *Smart tv's*, pode também, ser utilizado para os dispositivos móveis existentes, como os *smartphones*, *tablets*, *iphones* e *ipads*, em que estes, assim como as *Smart Tv's*, apresentam vários fabricantes diferentes, com diferentes sistemas operativos (várias versões de Android OS existentes no mercado), o que, assim como as *Smart Tv's*, exigia à empresa dispor de vários dispositivos diferentes no local de trabalho.

Em menor número, encontrávamos as *Smart Boxes e Gaming Systems*, isto porque a empresa apenas proporcionava suporte a alguns fabricantes (no caso das *Smart Boxes*, *Amazon Fire Tv, Android Nexus TV, Roku* e no caso dos *Gaming Systems*, *Playstation e Xbox*).

Relativamente às *Smart Boxes*, a quantidade de dispositivos variava de acordo com os fabricantes. No caso da *Amazon Fire TV e Android Nexus TV,* a empresa apenas possuía um dispositivo presente no local de trabalho, que ia sendo atualizada à medida que novas versões de software iam sendo lançadas para o mercado, não existindo necessidade de adquirir mais quantidade deste dispositivo, visto que, não existia outros modelos diferentes, nem outros modelos com funcionalidades diferentes.

Em relação às *Boxes* Roku, estas já se encontravam em maior quantidade no local de trabalho, visto existirem diferentes modelos no mercado, que, apesar de não

variarem muito a nível de funcionalidades, variavam sim, a nível de *hardware*, o que afetava imenso o *software* desenvolvido a nível de desempenho.

Em relação aos *Gaming Systems*, a situação acabada de mencionar para as *Smart Boxes Amazon Fire Tv* e *Android Nexus TV,* repetia-se, existindo apenas a necessidade de fornecer à equipa dois dispositivos (*Playstation 4 e Xbox One*), que se tratam dos modelos mais recentes destas plataformas, e, sendo apenas necessário ir atualizando o *software*, para se ter disponível, a versão mais recente.

Por último, mas não menos importante, a plataforma mais simples de se ter acesso, eram os *Web Browser's.* Estes são constituídos pelos já conhecidos *Google Chrome, Mozilla FireFox, Safari e Microsoft Edge*. Este eram utilizados diretamente nos computadores da equipa de *quality assurance*, assim como, também eram verificadas as versões *mobile* destes, utilizando para isso, os vários dispositivos existentes no local de trabalho. Esta verificação extra nos dispositivos *mobile* era efetuada, pois, na atualidade, os *Web Browsers* não se encontram limitados apenas a computadores, sendo possível também utilizá-los em dispositivos móveis, pelo que, era necessário garantir, o correto funcionamento dos *Web Sites* desenvolvidos em todas as versões dos *Web Browsers* existentes.

3. CONCLUSÃO E TRABALHO FUTURO

3.1.VISÃO CRÍTICA SOBRE O ESTÁGIO

3.1.1. Ambiente e cultura da empresa

No decorrer do ano 2011, é fundada a empresa ABC, com o intuito de se focar no desenvolvimento de software para *Smart Tv's*. Sendo esta, uma empresa relativamente recente e devido ao rápido crescimento que foi alvo no mercado de *software* direcionado para *Smart Tv's*, foi implementada uma cultura de crescimento na estrutura da empresa, na qual, o objetivo é tentar sempre evoluir e fazer sempre melhor, em cada *software* desenvolvido.

O ambiente que é possível encontrar, é, a nível geral, um bom ambiente, no qual se destaca a entreajuda, diálogo e bem-estar dos vários colaboradores. Para além disto, trata-se também de um ambiente que fornece excelentes oportunidades a todos os trabalhadores, desde do contacto com empresas de renome mundial para realização de projetos, assim como oportunidades para viajar e conhecer novos locais, sempre no âmbito de um projeto que a empresa irá abraçar. Como a empresa apresenta neste momento a sua mão de obra espalhada por 4 países, sendo estes a Inglaterra, Espanha, Estados Unidos e Portugal (Covilhã e Funchal), temos constante contacto com pessoas de diferentes países e culturas e este contacto é feito a nível diário de forma a coordenar todo o trabalho a ser realizado, isto permite, por exemplo melhorar o inglês, assim as relações com colegas de trabalho. Falando em especifico na filial do Funchal, é um excelente ambiente para se estagiar, pois a maioria dos trabalhadores estudaram quase todos na Universidade da Madeira, e em alguma fase desse período conheceram-se ou conhecem algum amigo em comum, o que cria um ambiente muito amigável, descontraído e de rápida integração.

3.1.2. Próximo passo – Processos Internos e Automação de testes

De forma à empresa ABC melhorar de forma significativa, melhorar os processos internos da empresa no que respeita ao processo de desenvolvimento, deveria estar no topo da lista de prioridades. Não me irei alongar sobre este assunto, pois acredito que ao longo do relatório, já foram descritos e apresentados, problemas e soluções sobre o atual processo de desenvolvimento da empresa.

Abordando agora outra área que a empresa poderia investir, de forma a melhorar o seu processo de *quality assurance*, esta poderia apostar em automação de testes.

É verdade que os testes manuais permitem encontram bastantes erros nas aplicações, pois é possível realizar testes de forma a abranger bastantes pormenores do software, mas acaba por tornar-se uma tarefa maçante e repetitiva. Nestas tarefas repetitivas, encontrámos casos de teste que se podem automatizar, de forma a serem rapidamente validados, permitindo assim automatizar a validação destes casos de teste.

A automação de testes é realizada a partir de software especifico que interpreta a automação criada de acordo com as pré-condições criadas, efetua os passos, verifica a validação de cada um e devolve ao utilizador um conjunto de resultados referentes a cada ação realizada.
Na automação de testes, existem 2 abordagens que podem ser tomadas: testes de interface gráfica e testes baseados em código.

Nos testes de interface gráfica, o software realiza uma gravação de toda a interação do utilizador com o software, registando todos as zonas clicadas e toda a informação introduzida. Após a gravação, o utilizador pode personalizar cada ação realizada, atribuindo tempos de espera entre cada ação ou tempos de espera entre a ação e o resultado, forma a melhor observar os resultados que vai obtendo.

Neste tipo de testes, o utilizador também pode criar o seu próprio script de teste utilizando todas as funções fornecidas pelo software de teste, em que cada função já está preparada para realizar uma determinada ação. Quando este método é usado, o

utilizador escolhe a ação que pretende realizar dentro do software, e depois seleciona com o rato no software a testar, qual a parte a que aquela ação se destina e assim sucessivamente até ter todas as ações pretendidas.

Após isto, apenas tem que executar o script, e o software de teste realiza todas as ações, permitindo ao utilizador ver estas todas a serem realizadas em tempo real. Caso alguma das ações não tenha o resultado esperado, esta fica marcada a vermelho para sinalizar ao utilizador a situação de insucesso, mas o software tenta, na mesma, realizar as restantes ações. Caso a ação seguinte dependa diretamente do sucesso da anterior, mas esta não consegui ser realizada, o script para automaticamente, sinalizando qual a ação que interrompeu o script e qual o motivo. Um exemplo disto, podia ser um script em que o utilizador após realizar login na sua conta de email, iria selecionar uma pasta "arquivados" que contém emails arquivados, mas ao executar o script, foi introduzido dados de login errados, marcando a ação de login a vermelho devido ao insucesso da mesma, e assim parando automaticamente o script pois este não consegue nenhuma pasta "arquivados".

Um exemplo em que a ação apesar de não ser completada com sucesso não afeta o restante script, pode ser, por exemplo, um script em que as ações definidas pelo utilizador apenas consistem em selecionar sucessivamente várias imagens existentes numa página web. Caso alguma destas seleções falhe, desde que a próxima seleção ocorra, o script continua a sua execução normal apenas sinalizando a vermelho a ação que não conseguiu selecionar a imagem alvo.

No que respeita aos testes baseados em código, estes são totalmente escritos em linguagens de programação, testando especificamente as várias classes, módulos e bibliotecas utilizadas no software alvo, utilizando argumentos de entrada e observando-se o comportamento de determinadas seções de código, observando se estas apresentam o comportamento esperado quando sujeitas a diversas circunstâncias.

Este tipo de testes é cada vez mais usual no processo de desenvolvimento de software, principal no desenvolvimento de software que que se estima que terá longa

vida no mercado, devido ao facto de que este pode apresentar quebras de funcionalidade quando é feito algum tipo de correção no código do software, por mais pequena que esta correção tenha sido

Para realização de testes de código, os scripts podem ser realizados em várias linguagens de programação, sendo C#, Perl, PHP, Ruby, Java e Python as mais populares e mais utilizadas pela maioria do software de testes e as que oferecerem mais variedade de *frameworks* para a criação de scripts. [33]

3.2.DESENVOLVIMENTO INTERPESSOAL E INTRAPESSOAL

3.2.1. Formação

Ao longo deste período, tive a oportunidade de melhorar bastantes aspetos pessoais assim como aspetos relacionados com quem me rodeia no dia a dia no local de trabalho.

A nível intrapessoal, melhorei bastante os meus conhecimentos de *quality assurance* graças ao trabalho realizado diariamente, assim como a qualidade dos testes que realizava. Aprendi como criar *test scripts* para a realização de testes, fui introduzido ao mundo da automação de testes, melhorei imenso na gestão do tempo disponível e na priorização das tarefas a realizar. Melhorei também o meu nível de inglês, escrito e oral, aprendi a lidar com pressão e com prazos para realizar as tarefas, tornando-me mais eficiente nestes aspetos, não deixando este afetar o meu trabalho.

A nível interpessoal, melhorei o meu relacionamento diário com os colegas de trabalho de forma a haver sempre um bom ambiente de trabalho e de forma a promover a interajuda. Aprendi a aceitar melhorar as opiniões e pontos de vista dos outros, assim como a "discutir" de forma saudável com os colegas de forma a melhorarmos algum aspeto de algum projeto.

3.2.2. Ferramentas e Tecnologias

Ao longo deste período de estágio, tive a oportunidade de lidar com diferentes ferramentas e tecnologias, sendo que, a nível de ferramentas, utilizei algumas que me eram totalmente desconhecidas, enquanto que, a nível de dispositivos e tecnologias, apenas algumas me eram totalmente desconhecidas.

Ao lidar diariamente com toda esta tecnologia, foi-me permitido impulsionar o meu nível conhecimento, o que permitiu, rapidamente, elevar o nível de qualidade relativo às tarefas desempenhadas.

3.3. OPINIÃO PESSOAL SOBRE O ESTÁGIO

É chegada a altura de poder refletir sobre esta experiência. De forma concisa, irei abordar o que considerei de relevante importância e consequente influencia no decorrer do estágio.

Como já foi abordado previamente, este estágio surge enquadrado no desempenhar das minhas funções enquanto *quality assurance*.

Estando já a desempenhar funções como *quality assurance* para a empresa selecionada para realização de estágio, não fui alvo de nenhum processo seletivo aquando da realização deste estágio., estando já familiarizado com a empresa, o que permitiu obter uma resposta positiva e sem obstáculos muita rapidamente por parte da direção para realização do estágio.

As condições de trabalho, na minha opinião são excelentes, tratando-se de um escritório com espaço para uma equipa de grandes dimensões (entre 25 - 30 colaboradores), no qual cada colaborador dispõe de um portátil, rato e monitor. A nível de conforto, os colaboradores dispõem de cadeiras bastantes confortáveis, ar-condicionado e local para refeições com micro-ondas, incluindo uma máquina de café. A localização também é excelente, estando localizado a 1 minuto do centro comercial La Vie, no centro do Funchal, o que permite aos colaboradores fácil acesso a estacionamento, restaurantes e departamentos do estado (finanças, segurança social e outros) caso algum colaborador tenha a necessidade de revolver alguma situação pessoal.

Em vez de secretárias individuais, foi optado por mesas longas que permitem aos colaboradores terem o seu espaço individual e, ao mesmo tempo, partilhar o espaço com os restantes colaboradores.

Destaco de forma importante, o ambiente existente no espaço de trabalho, tratando-se de um ambiente de entre ajuda, no qual *developers*, *designers, team leaders* e *quality assurance* partilham o mesmo espaço, permitindo a troca de ideias entre os diferentes departamentos e existindo sempre disponibilidade para ajudar a ultrapassar qualquer obstáculo encontrado. Este tipo de ambiente beneficia do facto de quase todos os colaboradores terem sido alunos da Universidade da Madeira,

permitindo que se conhecem-se já previamente, e tornado o processo de adaptação um processo extremamente simples e curto.

Apesar de todo este ambiente positivo, não posso deixar de referir a falta de processos em todo o processo de desenvolvimento de software por parte da empresa e a falta de formação adequada para o departamento de *quality assurance*, transmitindo uma ideia errada daquilo que é o papel de um *quality assurance* aos seus colaboradores.

Tendo em conta tudo o que foi mencionado anteriormente, considero que foi uma experiencia positiva, que me enriqueceu, quer a nível profissional como a nível pessoal, permitindo-me aplicar conhecimentos adquiridos a nível académico e melhorá-los e adquirir novas competências que poderão vir a tornar-se importantes para o meu futuro profissional.

De forma a finalizar esta retrospetiva, não poderia deixar de frisar, o quão importante considero o estágio profissional em cursos superiores, sendo que, na minha perspetiva, deveria haver um esforço por parte da Universidade para proporcionar mais oportunidades de estágio aos seus alunos, pois esta experiencia de estágio permite aos alunos terem um "sabor" do que realmente é "trabalhar" no âmbito de uma empresa, e permite também, perceber mais rapidamente que, todo o conhecimento teórico que se acumula ao longo dos vários anos de percurso académico, quando aplicado em situações reais do mercado de trabalho não se trata apenas de "fazer A para obter B", mas sim de adaptação constante às várias situações que se vai enfrentando no dia-a-dia no local de trabalho. Respeitando todas as opiniões, considero o estágio a opção mais enriquecedora e completa.

3.4.PERSPETIVAS FUTURAS

3.4.1. Empresa

Estamos perante um crescimento bastante elevado do mercado de *Smart Tv's*, e, consequentemente, também o mercado de aplicações para estes dispositivos se encontram em elevado crescimento, isto, graças ao estatuto de "acessório obrigatório" que estes dispositivos alcançaram, existindo, pelo menos um, na grande maioria das habitações da população.

Por sua vez, isto contribui para que a empresa ABC, graças às ferramentas de rápido desenvolvimento que possui, se consiga destacar, na área de desenvolvimento de aplicações para *Smart Tv's*, permitindo que esta continue a crescer, ano após ano, como tem vindo a acontecer desde da sua fundação.

De acordo com o que foi apresentado, a empresa apresenta todas as condições para continuar a crescer a bom ritmo e para continuar a destacar-se no desenvolvimento de software para *Smart Tv's*, estando apenas, condicionada a nível interno, pela forma como está a ser gerida, como foi relatado ao longo deste relatório.

3.4.2. Pessoais

Considerando todo o período de estágio realizado, e, ponderando toda a informação e conhecimento que foi adquirido ao longo desse mesmo período, devo confessar, que foi desperta em mim, a curiosidade sobre o mundo de *quality assurance*, tendo esta área se revelado, uma área na qual, o processo de adaptação ocorreu se forma natural, juntamente com um certo nível de facilidade, tendo em conta o pouco conhecimento que apresentava no inicio desta etapa, relativamente às funções que iria desempenhar.

Graças à fácil adaptação e ao interesse despertado, apercebi-me que, de livre e espontânea vontade, procurava mais informação e adquirir mais conhecimentos, relativamente a *quality assurance*, quer para melhoria das funções a desempenhar, quer

para satisfação e evolução a nível pessoal, demonstrando, a mim mesmo, o "gosto" que tinha desenvolvido pela área.

Ponderando todos os aspetos apresentados, tenciono, para o meu futuro, continuar a evoluir nesta área, quer a nível profissional como a nível pessoal, se oportunidades para tal, se presenciarem no meu futuro.

REFERÊNCIAS

[1] *History of Quality* - ASQ [Internet]. Asq.org. 2018 [acedido a 12 janeiro 2017]. Disponível em: http://asq.org/learn-about-quality/history-of-quality/overview/overview.html

[2] *Quality Assurance and Software Testing: A Brief History* [Internet]. Sauce Labs. [acedido a 5 janeiro 2017]. Disponível em: https://saucelabs.com/blog/quality-assurance-and-software-testing-a-brief-history

[3] *Farcic V. Black-box vs White-box Testing* [Internet]. Technology Conversations. [acedido a 20 março 2017]. Disponível em: https://technologyconversations.com/2013/12/11/black-box-vs-white-box-testing/

[4] *Software Testing Methodologies - Learn The Methods & Tools* [Internet]. Inflectra.com. [acedido a 25 julho 2017]. Disponível em: https://www.inflectra.com/ideas/topic/testing-methodologies.aspx

[5] *Agile Methodologies for Software Development* [Internet]. VersionOne. [acedido a 9 fevereiro 2017]. Disponível em: https://www.versionone.com/agile-101/agile-methodologies/

[6] *Bartlett J. Agile Testing: The Role of QA in Agile - TestLodge Blog* [Internet]. TestLodge Blog. [acedido a 13 maio 2017]. Disponível em: https://blog.testlodge.com/the-role-of-qa-in-agile/

[7] *QA in an Agile Environment | Intelliware Development* [Internet]. Intelliware Development Inc. [cited 30 junho 2017]. Disponível em: http://www.intelliware.com/qa-in-an-agile-environment/

[8] *Stark M. Quality Assurance for Agile Projects: What's Required?* [Internet]. iBeta Quality Assurance. [acedido a 23 abril 2017]. Disponível em: https://www.ibeta.com/qa-for-fast-paced-agile-projects/

[9] *Lean-Agile Software Development, Achieving Enterprise Agility*. 1st ed. Boston: Pearson Education, Inc; 2010.

[10] *Quality Assurance is broken. Here's how we can make it as agile as everything else.* [Internet]. freeCodeCamp. [acedido a 2 janeiro 2017]. Disponível em: https://medium.freecodecamp.org/quality-assurance-is-broken-heres-how-we-can-make-it-as-agile-as-everything-else-64bd19d5e426

[11] *A Practical Guide to implementing Agile QA process on Scrum Projects* [Internet]. www.scrumalliance.org. 2008 [acedido a 6 fevereiro 2018]. Disponível em: https://www.scrumalliance.org/system/resource_files/0000/0459/AgileQA.pdf

[12] *Selenium - Web Browser Automation* [Internet]. Seleniumhq.org. [acedido a 15 dezembro 2017]. Disponível em: http://www.seleniumhq.org/

[13] *Wiki: Welcome Visitors* [Internet]. Fit.c2.com. [acedido a 12 dezembro 2017]. Disponível em: http://fit.c2.com/

[14] *Wiki: Introduction To Fit* [Internet]. Fit.c2.com. [acedido a 13 dezembro 2017]. Disponível em: http://fit.c2.com/wiki.cgi?IntroductionToFit

[15] *Open source automated web testing tool* [Internet]. Wet.qantom.org. [acedido a 14 dezembro 2017]. Disponível em: http://wet.qantom.org/roadmap.html

[16] *Project W. Watir is...* [Internet]. Watir.com. [acedido a 14 dezembro 2017]. Disponível em: http://watir.com/

[17] *JUnit 5* [Internet]. Junit.org. [acedido a 14 dezembro 2017]. Disponível em: http://junit.org/

[18] *HttpUnit Home* [Internet]. Httpunit.sourceforge.net. [acedido a 14 dezembro 2017]. Disponível em: http://httpunit.sourceforge.net

[19] *The World's Most Popular API Testing Tool | SoapUI* [Internet]. Soapui.org. [acedido a 12 d 2017]. Disponível em: https://www.soapui.org/

[20] What is scrum? - A brief introduction | Atlassian [Internet]. Atlassian. [acedido a 12 janeiro 2018]. Disponível em: https://www.atlassian.com/agile/scrum

[21] Cohn M. User Stories and User Story Examples by Mike Cohn [Internet]. Mountain Goat Software. [acedido a 10 fevereiro 2017]. Disponível em: https://www.mountaingoatsoftware.com/agile/user-stories

[22] *Mantis Bug Tracker* [Internet]. Mantisbt.org. [acedido a 26 agosto 2017]. Disponível em: https://www.mantisbt.org/

[23] *Jira | Issue & Project Tracking Software | Atlassian* [Internet]. Atlassian. [acedido a 26 agosto 2017]. Disponível em: https://www.atlassian.com/software/jira

[24] *Replace all over the place with ONE place*. [Internet]. Basecamp.com. [acedido a 26 agosto 2017]. Disponível em: https://basecamp.com/

[15] *FileZilla - The free FTP solution* [Internet]. Filezilla-project.org. [acedido a 26 agosto 2017]. Disponível em: https://filezilla-project.org/

[26] *The Most Seamless Beta Distribution Experience for You and Your Testers*. | Beta by Crashlytics [Internet]. Try.crashlytics.com. [acedido a 26 agosto 2017]. Disponível em: http://try.crashlytics.com/beta/

[27] *Postman* [Internet]. Postman. [acedido a 26 agosto 2017]. Disponível em: https://www.getpostman.com/

[28] *SharePoint 2016 – Software de Colaboração em Equipa* [Internet]. Products.office.com. [acedido a 26 agosto 2017]. Disponível em: https://products.office.com/pt-pt/sharepoint/collaboration

[29] *Dropbox* [Internet]. Dropbox. [acedido a 26 agosto 2017]. Disponível em: https://www.dropbox.com/

[30] *Bem-vindo ao Microsoft OneDrive* [Internet]. Onedrive.live.com. [acedido a 26 agosto 2017]. Disponível em: https://onedrive.live.com/about/pt-pt/

[31] *Descarregar o Skype* | Chamadas gratuitas | Aplicação de conversa [Internet]. Skype.com. [acedido a 26 agosto 2017]. Disponível em: https://www.skype.com/pt/get-skype/

[32] *Microsoft Teams – Group Chat software* [Internet]. Products.office.com. [acedido a 26 agosto 2017]. Disponível em: https://products.office.com/en-us/microsoft-teams/group-chat-software

[33] *Automação de Testes: Porquê, o Quê e Como? - SQA* [Internet]. [acedido a 28 novembro 2017]. Disponível em: http://www.linkconsulting.com/sqa/automacao-de-testes-porque-o-que-e-como/

ANEXOS

Anexo A - Organograma Geral da Empresa ABC

- Figura 1 - Organograma da empresa ABC