IPV - ESTGV |

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Dedicatória

Dedico esta dissertação aos meus pais, Maria Alice Neves Ferreira Trigo e Simão Cláudio Alves Trigo, por terem insistido para me inscrever no curso de Mestrado. Dedico ainda à minha irmã, Francisca Miguel Ferreira Trigo, por me ter ajudado e apoiado durante todo o percurso universitário. Dedico ao meu avô Manuel Ferreira Neves avô que "partiu" a meio desta viagem, mas que onde quer que esteja está a dar-me força e coragem para continuar. E por último, ao meu namorado, Rafael, que tem também apoiado e incentivado imenso esta fase da minha vida.

Agradecimentos

Ao Professor Valter Alves pela sua orientação e sabedoria durante o período de realização da tese.

Aos quatro sócios da empresa: Rui Pinheiro, Carlos Sampaio, Gonçalo Rijo e Hugo Cristina pela boa disposição e simpatia de todos e por todo o conhecimento que transmitiram.

Aos meus pais, irmã e toda a família por apoiaram sempre as minhas escolhas e por me terem proporcionado esta oportunidade de crescer pessoal e profissionalmente.

Ao meu avô que "partiu" a meio desta minha viagem, deixando-me como herança a paz, a serenidade, a retidão, os ensinamentos, os valores e os princípios morais que me engrandecem enquanto ser humano.

Aos meus amigos, em especial Marta Gonçalves, Carina Penedo, Sérgio Machado, Rui Ferrão e Rafael Ribeiro, pela amizade e o apoio moral que me foi dado nesta altura da minha vida.

À Vera Pina pelo apoio moral, pela preocupação, pelo reforço positivo e incentivo para não desistir e também pelo conhecimento transmito na altura da realização desta dissertação.

Ao Rafael por estar sempre presente na partilha dos bons e maus momentos da nossa vida, pelo apoio, ânimo, encorajamento e incentivo com que me presenteou neste momento excecionalmente importante.

Resumo

A principal razão que leva a mestranda a adotar este projeto deve-se ao facto de ser uma *tester* de *software* no âmbito da sua atividade profissional então, decidiu aprofundar o conhecimento sobre esta temática, investigando o impacto da realização de testes de *software* dentro de uma empresa.

O principal objetivo é adquirir os conhecimentos necessários para a mestranda se tornar numa referência ao nível dos testes de *software* e controlo da qualidade, para isso pretende efetuar o levantamento de alguns tipos de testes de *software* tais como: testes aplicacionais, testes funcionais, testes unitários e testes através da especificação, saber o porquê de uma boa documentação, quais as boas práticas no âmbito dos testes de *software* e qual o impacto da realização na empresa.

Para a concretização desta dissertação é fundamental efetuar o levantamento sobre os tipos de testes de *software* referidos anteriormente, evidenciar a importância da documentação e demonstrar qual o impacto destes testes dentro de uma empresa.

De forma a garantir que os testes tenham sucesso, é necessário assegurar que é feita uma análise prévia a cada teste de forma a garantir que o seu resultado tenha sucesso.

Um teste de *software* é um processo que implica encontrar vulnerabilidades. Para que este seja fidedigno e eficaz deve ser realizado por um *tester*, atendo ao facto de que quem desenvolveu o código está ligado ao seu trabalho, portanto poderá não ser eficaz na realização do teste,

É de primordial importância efetuar a documentação necessária para os testes que são feitos a um determinado *software*. Assim, se existe alguma dúvida, bastará efetuar uma consulta à documentação realizada pelo *tester* e assim evita que seja cometido algum erro ou a repetição de novos testes de *software*.

Após cada iteração dos testes, deverão ser produzidas conclusões sobre o que foi feito, para se perceber se o resultado foi o que se estava à espera e se o teste foi feito de acordo com a especificação descrita detalhadamente antes de se iniciar determinada tarefa.

Índice

DEDICATÓRIA V
AGRADECIMENTOS VII
RESUMO IX
ÍNDICE XI
ÍNDICE DE FIGURAS XIII
ÍNDICE DE TABELAS XVII
GLOSSÁRIO XIX
ABREVIATURAS XXI
1. INTRODUÇÃO 1
1.1 Motivação 4
1.2 Objetivo 4
1.3 Metodologia 4
2. REVISÃO DA LITERATURA 7
2.1 Testes de software 7
2.1.1 Pertinência dos testes 9
2.1.2 Abordagem estratégica ao teste de software 10
2.1.3 Principais testes de software 14
2.2 Testes Aplicacionais 15
2.2.1 Paradigmas dos Testes aplicacionais 16
2.3 Testes Funcionais 20
2.4 Testes unitários 22
2.4.3 *Code Coverage* 25
2.5 Testes através da especificação 26
2.5.1 Requisitos Funcionais e Não Funcionais 31
2.5.2. Técnicas para levantamento de requisitos 32
2.6 Importância da documentação 34
3. ESTUDO DE CASO 37
3.1 Tickets 37
3.1.1 Gestão de tickets 38
3.1.2 Estados dos tickets 39

3.1.4 Prioridades dos Tickets 41
3.2 Documentação dentro da equipa *43*
3.4 Ambientes *50*
3.4.1 Ambiente DEV 50
3.4.2 Ambiente PRE-QUA 53
3.4.3 Ambiente QUA 54
3.4.4 Ambiente PRD 54
3.5 Elaboração dos testes *55*
3.5.1 Reprodução do erro reportado 56
3.5.2 Testes aos *tickets* resolvidos 56
3.6 Análise do impacto dos testes *60*
4. CONCLUSÃO 67
5. REFERÊNCIAS 69
ANEXO A: TRABALHOS REALIZADOS 75
ANEXO B: *TICKETS* 77
B.1 Numero de tickets num deploy *77*
B.2 Número de tickets aprovados sem tester *78*
B.3 Número de tickets aprovados com tester *78*

Índice de Figuras

Figura 1 - Defeito, Erro, Falha Adaptado: (Neto, 2014) 9

Figura 2 - Etapas dos Testes de Verificação Adaptado: (Bartié, 2002) 12

Figura 3 - Estratégia de Teste Adaptado: (Sousa, 2015) 13

Figura 4 - Processo de teste de software Adaptado: (Eliza & Lagares, 2017) 14

Figura 5 - Testes Aplicacionais Adaptado: (Software Application Testing, 2013)....16

Figura 6 - Teste Caixa Branca Adaptado: (Rongala, 2015) 17

Figura 7 - Teste Caixa Preta Adaptado: (Patil, 2017)....18

Figura 8 - Teste Caixa Cinzenta Adaptado: (Basic software testing v2.20, 2015)....19

Figura 9 - Teste Funcional Adaptado: (Lopes, 2009)....21

Figura 10 - Estratégia do Teste Funcional Adaptado: (Lopes, 2009)....21

Figura 11 - Exemplo de um Teste Unitário (Tercete, 2013) 23

Figura 12 - Testes Unitários 23

Figura 13 - Ciclo de um teste unitário. Adaptado (Bahia, 2015): 24

Figura 14 - Análise do code coverage 25

Figura 15 - Teste Unitário (Farias, 2017) 26

Figura 16 - Especificação de Requisitos e Especificação Foram Adaptado: (Figueiredo, Neve, Magalhães, & Pinto, 2002)....27

Figura 17 - Especificação Formal – Métodos Adaptado: (Carreira, 2014) 28

Figura 18 - Especificação Formal Adaptado: (Andrade, 2016)....30

Figura 19 - Definição de requisito Adaptado: (crvs-dgb, s.d.) 31

Figura 20 - Requisito Funcional Adaptado: (Ventura , 2016) 32

Figura 21 - Levantamento e analise de requisitos Adaptado: (Benadi, 2017) 34

Figura 22 - Histórico dos Tickets 38

Figura 23 - Gestão de tickets 38

Figura 24 - Estado dos tickets 39

Figura 25 - Ticket Urgente 41

Figura 26 - Grau de Prioridade dos Tickets 43

Figura 27 - Gestão dos Estados e Ambientes 45

Figura 28 - Tickets All Done 46

Figura 29 - Copiar dados da tabela 46

Figura 30 - Constituição de uma especificação 48

Figura 31 - Exemplo de commit 50

Figura 32 - Fazer Sync 51

Figura 33 - Pasta dos Re-Runnables 51

Figura 34 - Correr os re-runnables 52

Figura 35 - Fazer update-database 52

Figura 36 - Ambiente DEV 53

Figura 37 - Ambiente PRE-QUA 53

Figura 38 - Ambiente de QUA 54

Figura 39- Ambiente PRD 55

Figura 40 - Exemplo de um ticket 56

Figura 41 Criação de um novo ticket...58

Figura 42 -Ficheiro com o conteúdo de QUA 59

Figura 43 - Aguardar deploy em QUA 59

Figura 44 - Mensagem de atribuição e validação por parte dos stakeholders 60

Figura 45 – Voltaram para trás 61

Figura 46 - Ticket que voltou para trás (exemplo) 61

Figura 47 - Tickets que voltaram para trás 62

Figura 48 - Tickets reprovados 63

Figura 49 - Número de tickets 64

Figura 50 - Número de tickets em cada deploy 77

Figura 51 - Número de tickets aprovados antes 78

Figura 52 - Tickets Aprovados 79

Índice de Tabelas

Tabela 1 - Vantagens e Desvantagens dos Teste de Caixa Branca 17

Tabela 2 - Vantagens e Desvantagens do Teste Caixa Preta..18

Tabela 3 - Vantagens e Desvantagens do Teste Caixa Cinzenta....................................19

Tabela 4 - Vantagens e Desvantagens dos Testes Funcionais..22

Tabela 5 - Vantagens e Desvantagens dos Testes Unitários 25

Tabela 6 - Vantagens e Desvantagens da Especificação Formal....................................28

Tabela 7 - Gestão de Tarefas 44

Tabela 8 - Fecho de tickets..47

Tabela 9 - Filtro fecho de tickets 48

Tabela 10 - Histórico de Versões de uma Especificação Funcional (exemplo) 49

Tabela 11 - Filtro do tester 57

Tabela 12 - Procedimento de teste...57

Glossário

Bugs - erros que acontecem na aplicação.

Caso de teste - conjunto de *inputs* de teste, contextos de execução e resultados esperados, desenvolvido para atingir um determinado objetivo.

Caso de uso - unidade funcional coerente provida pelo sistema, subsistema, ou classe manifestada por sequências de mensagens intercambiáveis entre os sistemas e um ou mais atores.

Debug - O comando é tido como especialmente útil para interpretar e monitorar o funcionamento de programas executáveis, bem como encontrar possíveis erros operacionais.

Deploy - atualização da base de código ou base de dados num dos ambientes.

Layout- modo de distribuição dos elementos gráficos em determinado espaço.

Refactoring – é o processo de modificar um sistema de *software* para melhorar a estrutura interna do código sem alterar seu comportamento externo.

Abreviaturas

- DEV – Ambiente local (Alpha*)*
- PRE – QUA – Ambiente de pré-qualidade
- QUA – Ambiente de qualidade (Beta*)*
- PRD – Ambiente de produção (Release*)*

1. Introdução

Esta dissertação aborda a temática dos testes de *software,* nomeadamente em termos da qualidade do *software* desenvolvido. De forma a centralizar o tema da dissertação num âmbito mais específico, irá ser feito uma reflexão sobre o impacto da realização dos testes de *software,* numa empresa cliente, que opera a gestão de resíduos da *Estamos Juntos*, onde a orientada se encontra a estagiar.

A Estamos Juntos define-se como "uma empresa portuguesa de intervenção internacional. A [sua] missão é a criação de soluções para os nossos clientes através da entrega e acompanhamento de projetos de tecnologias de informação e gestão organizacional com grande qualidade. O [seu] compromisso com os nossos clientes está para além do trabalho ou do projeto. A satisfação e o sucesso são atingidos através de um envolvimento total, de verdadeira parceria, estando ao lado do cliente para resolver problemas atuais e encontrar soluções para o futuro. Estamos juntos é mais que um nome. É verdadeiramente a nossa filosofia e postura no mercado." (estamosjuntos, s.d.)

Este tema surgiu dado que a mestranda se encontra a estagiar na empresa Estamos Juntos, tendo como principal função a elaboração e realização de testes e documentação de *software.*

Um teste de *software* é um método que valida a execução do programa de uma maneira controlada com o objetivo de avaliar o seu comportamento. A atividade de um teste exige conhecimento, planeamento, projeto, execução, acompanhamento, recursos e também uma grande interação entre equipas. Ao realizar um teste de *software* existem algumas dificuldades, tais como: ser um processo caro, falta de conhecimento sobre a relação custo e beneficio do teste, falta de profissionais na área de teste, não existe o conhecimento de técnicas de testes adequadas e preocupação com a atividade de teste na fase final do projeto (Crespo, 2004).

O teste de *software* exibe a atividade que elabora o processo de verificação e validação de software, que é a técnica mais utilizada no âmbito da garantia e da confiabilidade de *software* (Barbosa, 2009).

A área de testes de *software* é essencial para qualquer equipa de desenvolvimento moderna.

Nas aplicações informáticas modernas, temos quartos principais ambientes de desenvolvimento:

- Local (*Alpha*) – Este ambiente encontra-se nos computadores dos elementos da equipa e é o ambiente mais volátil de todos, o código encontra-se em constante mutação;

- Pré-Qualidade - Este ambiente é dedicado aos testes e é uma replicada do ambiente de qualidade, onde se pode estragar e sem intervir com os testes que os *stakeholders* fazem.

- Qualidade (*Beta*) – Este ambiente é mais controlado e tipicamente encontra-se num servidor externo em que podem existir acessos por parte de *stakeholders* e elementos exteriores à equipa de desenvolvimento;

- Produção (*Release*)– Este ambiente é o menos volátil e apenas contém código validado nos anteriores. É o ambiente que está exposto ao público, logo é desejável que tenha o mínimo de erros possível.

A uma atualização da base de código ou base de dados num dos ambientes, dá-se o nome de *deploy*.

Os programadores fazem o desenvolvimento que consta numa especificação que contem os requisitos e o *tester* testa o código, tendo em conta cada requisito, garantindo que o mesmo não vai para o ambiente de produção com erros. O programa que corre em ambiente de produção não tem de estar necessariamente a aparecer no ecrã. O que é passado para o ecrã é apenas uma camada de apresentação, só parte da informação é apresentada num ecrã onde este *software* terá de passar por alguns testes para haver garantia de que tudo se encontra sem erros. Numa primeira fase, o código é testado no ambiente local, posteriormente é efetuado o *deploy* no ambiente de qualidade onde são feitos testes mais exaustivos e validadas as funcionalidades, por fim é feito *deploy* em ambiente de produção.

No âmbito desta dissertação, irão ser abordadas com maior preponderância os seguintes tipos de testes de software: testes aplicacionais, funcionais, unitários e testes através da

especificação.

Os **testes aplicacionais** assumem-se como um elemento de elevada importância no ciclo de vida de um *software*, devido ao aumento de qualidade adequado nas aplicações e consequente redução de custos com manutenções futuras. (Jesus, 2013).

Os **testes funcionais** são testes que avaliam o comportamento da aplicação, são fornecidos os dados de entrada e o teste é executado, o resultado obtido é comparado ao resultado esperado. Este tipo de teste é aplicável a todas as fases do teste, sendo testadas as funcionalidades, os requerimentos, as regras de negócio presentes da documentação para efeitos de teste.

Os **testes unitários** são uma forma de testar unidades individuais de código fonte, unidades que podem ser métodos, classes, funcionalidades ou módulos.

O **teste através da especificação** consiste em especificar o objetivo e determinar se o programa satisfaz os requisitos funcionais e não funcionais que foram especificados. Entretanto, os critérios de teste, baseados em especificação, podem ser utilizados em qualquer fase de teste sem a necessidade de modificação (Maldonado, 2004).

O trabalho de um *tester* consiste na realização de testes de *software* adequados ao projeto e também na elaboração de documentação importante para o futuro do mesmo, como por exemplo, grelhas de teste, que serão preenchidas posteriormente, após a conclusão dos testes.

Foi efetuada uma pesquisa do estado da arte na área dos testes de *software* de forma a poder avaliar que áreas necessitavam de demonstração científica, e que temas iam ao encontro das áreas de interesse da orientada.

Segundo Mendes (2016):

> "*Ao testar um sistema, deve-se ter em conta a atividade que o teste assegura e deve responder às seguintes perguntas: Quais os atributos da qualidade que deverão ser testados? Quem realizará os testes? Que recursos serão utilizados? Quais as dependências entre os atributos de qualidade? Quais as dependências entre as atividades de desenvolvimento? Como o processo e a qualidade do sistema de software serão acompanhados?"*

1.1 Motivação

O tema desta dissertação é relevante do ponto de vista da mestranda, porque aborda uma temática em discussão no seu local de trabalho. Na sua atividade profissional efetua diariamente testes de *software* em aplicações informáticas da empresa, o seu principal foco de trabalho consiste em descobrir se as tarefas sobre um determinado requisito especificado estão a funcionar corretamente e não possuem *bugs*.

O conhecimento adquirido no âmbito desta dissertação será uma mais valia a nível profissional e a nível pessoal.

1.2 Objetivo

O objetivo desta dissertação passa por fazer uma revisão da literatura relativa á realização de testes de *software*. Complementarmente e no sentido de averiguar a pertencia deste tipo de testes, nomeadamente no caso especifico da empresa onde a orientada se encontra a trabalhar, pretende-se refletir sobre o impacto da atividade que a mestranda desempenha nessa empresa, e que consiste em verificar o software e reportar os erros.

No seio da equipa, sabe-se que a cada iteração do desenvolvimento do código há um processo de teste, sendo assim, os programadores delegam a tarefa dos testes do código no *tester*, sendo que, após serem feitos os testes e caso contenham erros, os programadores já têm a informação necessária para os corrigir.

1.3 Metodologia

A metodologia utilizada nesta dissertação de mestrado consiste em revisão bibliográfica complementada com análise de impacto da realização de um estudo de caso na empresa em que a mestranda se encontra a trabalhar.

Segundo Robert Yin (2013):

> *"O estudo de caso é uma investigação empírica que investiga um fenómeno*

contemporâneo em profundidade e no seu contexto da vida real especialmente quando os limites entre o fenómeno e contexto não claramente evidentes "

Este estudo de caso é explanatório porque tem como objetivo procurar evidências de que a entrada de um *tester* produz benefícios positivos na equipa, na medida que, ao haver uma bateria de testes mais especializados são detetados erros antes de passar aos ambientes.

Este caso de estudo tem uma tipologia única na medida em que, é apenas investigado o caso numa única equipa da empresa. De acordo com a finalidade do caso de estudo estamos perante um caso de estudo intrínseco porque o objeto de estudo é de interesse para a mestranda, visto que, esta pretende obter uma melhor compreensão sobre o caso apenas pelo interesse com o mesmo.

2. Revisão da Literatura

Nesta secção será feito um estudo dos conceitos teóricos relevantes no âmbito deste trabalho. Irá ser demonstrado o que são testes de software, testes aplicacionais e os seus paradigmas, testes funcionais, testes unitários, testes através da especificação e a importância da documentação dentro de uma equipa. Este capítulo tem como objetivo fazer um enquadramento dos conceitos teóricos que irão ser aplicados no estudo de caso.

2.1 Testes de software

Os testes de *software* surgiram nas décadas de 1960 e 1970 e eram elaborados pelas equipas de desenvolvimento. Mais tarde, a partir dos anos 1980, passou a ser um processo formal e, nos tempos de hoje, são uma parte fundamental e altamente importante no processo de desenvolvimento do *software* (Costa C. , 2001).

O teste de *software* não é algo novo, as empresas crescem sistemas sem uma metodologia com base na qualidade. Quando se compra um carro, este passou por diversos testes, quando é fabricado um brinquedo, este passa por testes, para saber se a tinta não é prejudicial às crianças e assim por diante. Todos querem produtos de qualidade, então, por que razão isso não é aplicado ao *Software*? (Gomes, 2014/2015).

Um teste é um processo que implica encontrar vulnerabilidades. Por isso, quem desenvolveu o código não deverá realizar o teste, pois vínculo emocional ao seu trabalho, pode pôr em causa sua eficácia (Pereira, 2011).

Um propósito fundamental para se fazer testes é a verificação das especificações. A **especificação** é a descrição do que o *software* deve fazer a partir daquilo que foi analisado anteriormente, onde é apresentada a solução de como os problemas levantados na análise devem ser resolvidos em desenvolvimento. A especificação é a forma de mensagem direta entre o cliente e a equipa de desenvolvimento (Tonsing, 2008).

Em primeiro lugar, realizam-se testes para verificar se tudo aquilo que foi especificado para um produto ou sistema se encontra como o pré-estabelecido, se o desempenho está funcional e se a implementação obedece ao que foi solicitado (Meriat, 2013).

As atividades de teste não se resumem à execução do mesmo, é necessário efetuar um planeamento, escolher as condições de teste e elaborar critérios de avaliação do teste.

Depois disso, é elaborado o teste e retiradas as devidas conclusões ou resultados. É necessário que o processo de teste seja transparente, desde o seu planeamento até à apresentação dos resultados, sendo que o *tester* deve contribuir para essa transparência, mantendo todas as partes envolvidas informadas sobre todo o processo de teste.

O processo de teste engloba uma estruturação de etapas, atividades, artefactos, papéis e responsabilidades que procuram a padronização dos trabalhos, gestão e controlo dos projetos de teste. O plano de testes, contém os objetivos e metas de teste, enquanto o projeto de testes detalha e especifica como o plano de testes irá ser executado. O caso de teste descreve as situações que devem ser testadas (Silva W. , 2012).

Falar de testes de *software* leva a um debate de conceitos relacionados com essa atividade,

> é preciso esclarecer a diferença entre defeitos, erros e falhas. **Defeito** é resultante de um ato inconsciente cometido ao tentar entender certa informação, resolver um problema ou usar o método ou uma ferramenta. **Erro** é uma revelação concreta de um defeito num *software* onde é vista a diferença entre o valor obtido e o esperado. **Falhas** é um comportamento operacional do *software* diferente do esperado pelo utilizador, que pode ser causada por múltiplos erros e alguns podem não causar uma falha. Ao observar a Figura 1 pode-se ver que há uma diferença entre os conceitos, defeitos podem equilibrar a revelação de erros num determinando produto, ou seja, neste caso a construção de um *software* é realizada de forma diferente no que se encontra especificado. Por fim, os erros geram falhas, ou seja, comportamentos inesperados de um *software,* afetam o utilizador final da aplicação podendo inviabilizar a sua utilização (Neto, 2014).

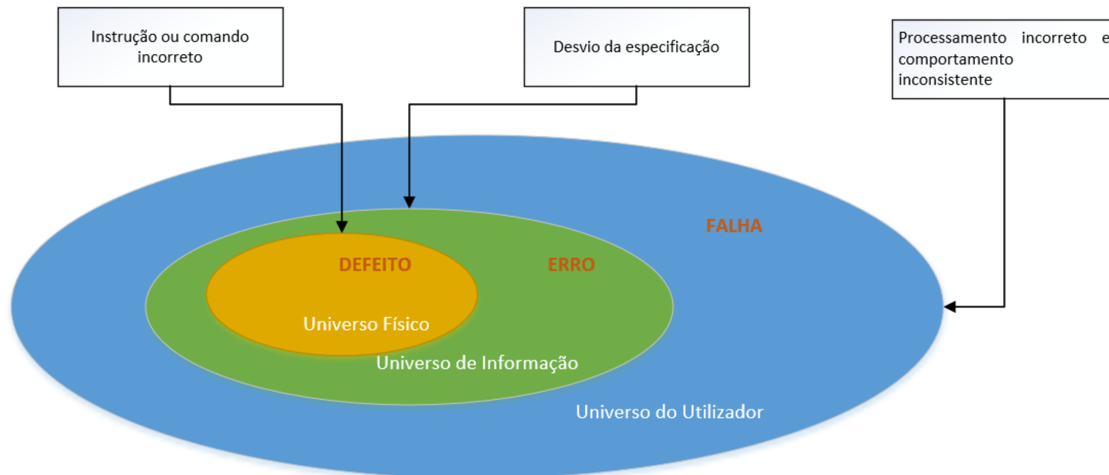


Figura 1 - Defeito, Erro, Falha Adaptado: (Neto, 2014)

2.1.1 Pertinência dos testes

Há vários fatores que justificam a realização de testes de *software*, no entanto, os principais fatores que levam os gestores de Tecnologias de Informação (TI) à realização de testes de software são (Carvalho, 2010):

- Diminuir o risco;
- Diminuir os custos associados à produção e manutenção;
- Garantir que todas as especificações e expectativas definidas e esperadas pelo cliente são cumpridas, para que não exista a insatisfação por parte do cliente;

Um fator muito importante para a realização dos testes de *software* prende-se, cada vez mais, com o facto de as organizações dependerem de mais sistemas que suportam os seus processos de negócio e, para além disso, os sistemas de informação estão cada vez mais complexos, incluindo muitos componentes. (Carvalho, 2010).

Além dos benefícios da prática de testes de *software,* outros benefícios práticos importantes são (Carvalho, 2010):

- Otimização do processo de desenvolvimento;
- Melhoria da qualidade no desenvolvimento, ou seja, existem requisitos,

implementação e documentação especifica;

- Garantia de entrega dos requisitos;
- Imposição de critérios de qualidade bem definidos e claros aos fornecedores de software;
- Cliente mais satisfeito;

2.1.2 Abordagem estratégica ao teste de software

Existem várias estratégias de teste de *software*, onde o principal objetivo é o fornecimento de diretrizes para a verificação, validação, organização e estratégia dos testes de *software*.

Um **teste de verificação** tem como objetivo garantir a qualidade de software, permitindo a verificação e validação. A **verificação** tem como objetivo garantir que o programa está a ser implementado corretamente, a questão que se coloca é: *"Estamos a construir bem o produto?"* e a **validação** tem como objetivo garantir que o programa se adequa aos requisitos, visando responder à pergunta *"Estamos a construir o produto correto?"* (Ferreira, 2010).

De seguida, serão mostrados os erros mais comuns que muitas vezes dificultam o processo de desenvolvimento da equipa:

- Incapacidade de entendimento das necessidades do utilizador;
- Engano na gestão do tamanho do projeto;
- Falta no planeamento de projeto;
- Falta de testes do produto final;
- Execução enganada do produto;
- Pouco incentivo à equipa de desenvolvimento;
- Atraso do plano do projeto;

- Mudança de ferramentas no inicio de desenvolvimento do software;
- Por vezes os programadores tentam eliminar tarefas necessárias a fim de encurtar o plano do projeto;
- Incapacidade de gestão do projeto;
- Falta de teste de unidade;
- Programadores cansados;
- Falta de tratamento de erros;
- Implementação incorreta de hardware;
- Falta de coordenação do processo de desenvolvimento de software;
- Execução de operações de base de dados na camada de aplicação, em vez de na camada de base de dados;
- Dados inválidos.

De seguida apresentam-se alguns exemplos de erros clássicos que ocorreram ao longo da história (Farias, 2017):

- O jogo do Rei Leão da Disney onde o que faltou foi um teste de multiplataformas;
- Ariane 5 que o faltou foi um teste de valores de limites;

Além de garantir a qualidade do software, segundo Bartié, **teste de verificação** é entendido como um processo de auditoria de atividades e avaliação de documentos produzidos durante o processo de *software*. Normalmente, as verificações são aplicadas a todos os produtos, ou seja, documentos, gráficos, manuais e código fonte elaborados em cada etapa do processo. Assim, evitar-se-ão dúvidas ou assuntos mal resolvidos que surgem e sigam para uma nova fase (Bartié, 2002).

A principal característica destes testes é o facto de não envolverem o processo de *software*, as atividades antecedem a criação da aplicação, assim poder-se-á garantir que as decisões e definições foram entendidas e aceites pelos diversos grupos que fazem parte do processo de

desenvolvimento (Bartié, 2002).

Para a realização deste teste existem as seguintes etapas, ver Figura 2:

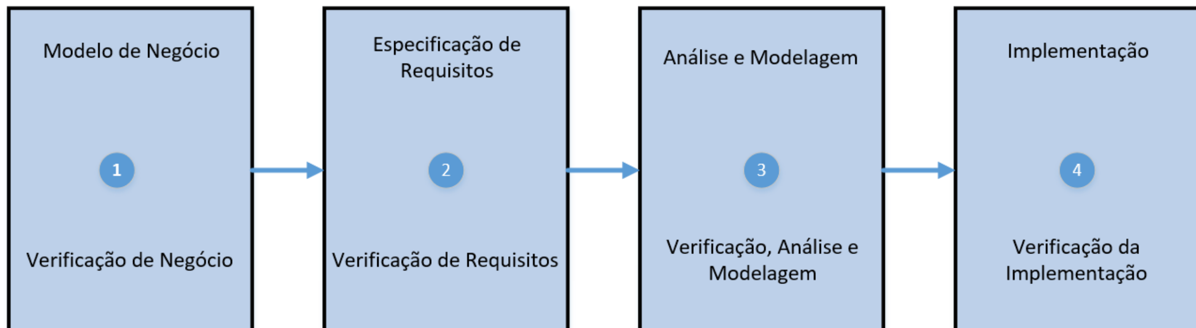


Figura 2 - Etapas dos Testes de Verificação Adaptado: (Bartié, 2002)

- Verificação do negócio;
- Verificação dos requisitos;
- Verificação, Análise e Modelagem;
- Verificação da Implementação;

Os **testes de validação** são entendidos como um método formal de avaliação de produtos tecnológicos onde podem ser aplicados componentes isolados, módulos existentes ou mesmo nos sistemas como um todo (Bartié, 2002).

O objetivo deste tipo de teste é avaliar a conformidade do *software* desenvolvido com os requisitos e as especificações analisadas nas etapas iniciais dos projetos (Bartié, 2002).

Um teste de *software* deve ter uma organização, ou seja, reconhece existir um trabalho contínuo no projeto, em conjunto com os vários grupos de trabalho, ou seja, programadores e grupo de teste entre outros, com a finalidade de garantir a qualidade de *software*. O programador é o responsável por escrever o código fonte do *software* e realizar testes unitários. O grupo independente de teste deverá testar o *software* e identificar os problemas associados à aplicação. O gestor de projeto deverá trabalhar com os programadores e o grupo independente de teste, para garantir que estão a ser efetuados testes rigorosos á aplicação (Ferreira, 2010).

Devem ser testados individualmente os componentes que integram a aplicação, a fim de verificar o objetivo de cada um e garantir que esta se encontra a funcionar adequadamente. Em seguida, devem-se reunir os diversos componentes que formam o *software*, permitindo fazer os respetivos testes de integração e validar se os requisitos estão a ser cumpridos (Ferreira, 2010).

Todos estes testes referidos anteriormente fazem parte da estratégia de teste como está ilustrado na Figura 3.

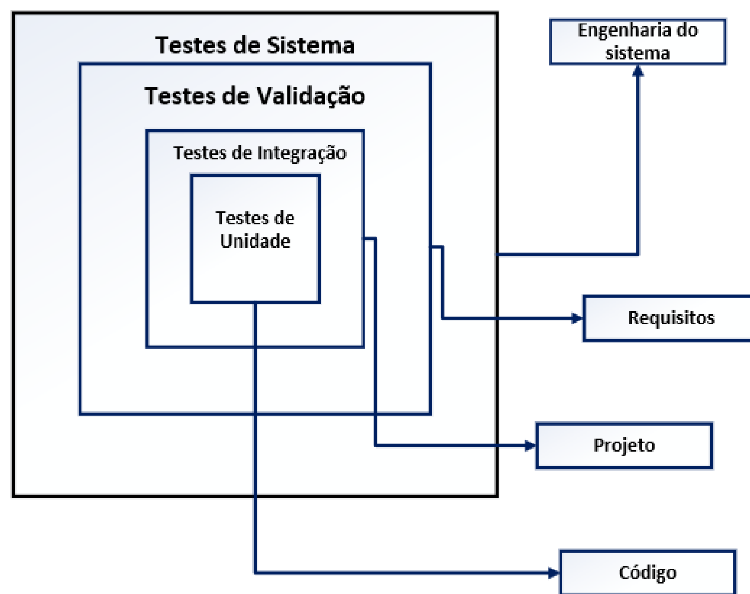


Figura 3 - Estratégia de Teste Adaptado: (Sousa, 2015)

Um **processo de teste de software** é um conjunto de passos constituídos por atividades, métodos e práticas, utilizadas para testar um *software*. Um processo de teste de *software* possui como subprocessos o planeamento, projeto, execução e registo do teste (Junior, 2007), ver Figura 4.

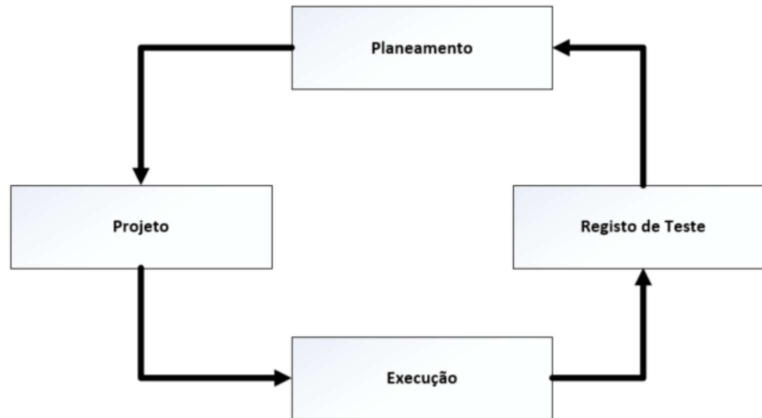


Figura 4 - Processo de teste de software Adaptado: (Eliza & Lagares, 2017)

2.1.3 Principais testes de software

Quando falamos em testes de *software* deve-se ter em conta que estes são divididos em diversos tipos, de acordo com seu objetivo particular. Para clarificar os conceitos irão ser apresentados de seguida os principais testes de *software* (Trust , 2015):

- **Teste de Configuração:** testa se o *software* funciona no *hardware* instalado;
- **Teste de Instalação:** testa se o *software* está instalado como planeado em diferentes *hardwares* e em diferentes condições, como pouco espaço de memória, interrupções de rede e interrupções durante a instalação;
- **Teste de Integridade:** testa a robustez do *software*;
- **Teste de Segurança:** testa se o sistema e os dados são acedidos de maneira segura;
- **Teste Funcional:** testa os requisitos funcionais e os casos de uso;
- **Teste Unitário**: testa uma componente isolada como uma classe do sistema;
- **Teste de Integração:** testa se um ou mais componentes funcionam de maneira satisfatória.
- **Teste de Volume:** testa o comportamento do sistema com um dado volume de dados e as suas transações durante um largo intervalo de tempo;

- **Teste de Desempenho:** este teste divide-se em três tipos, ou seja, teste de carga, teste de stress e teste de estabilidade;

- **Teste de Usabilidade:** este teste é focado na experiência do utilizador, na consistência da interface, *layout* e acesso as funcionalidades;

- **Testes de Caixa Branca e Caixa Preta:** o teste de caixa branca consiste em testar partes isoladas do código e o teste de caixa preta consiste em testar as funcionalidades não considerando a sua implementação (código);

- **Teste de Regressão:** faz-se um novo teste às componentes de modo a verificar se existe alguma modificação recente que originou algum efeito indesejado;

- **Teste de Manutenção**: testa se a mudança feita no ambiente não interferiu com o funcionamento da aplicação;

No que se refere ao seu âmbito os testes podem ser classificados como aplicacionais, funcionais, unitários ou por especificação, como se desenvolve nas secções seguintes.

2.2 Testes Aplicacionais

Os testes aplicacionais incluem tarefas de revisão como qualidade e configuração, monitorização de desempenho, simulação, estudo de viabilidade, revisão de documentos, revisão da base de dados, análise de algoritmos e execução de testes aplicacionais (Jesus, 2013).

Os testes aplicacionais são executados pelos programadores, depois que uma aplicação é desenvolvida ou antes da sua disponibilização para os utilizadores/clientes, ver Figura 5.

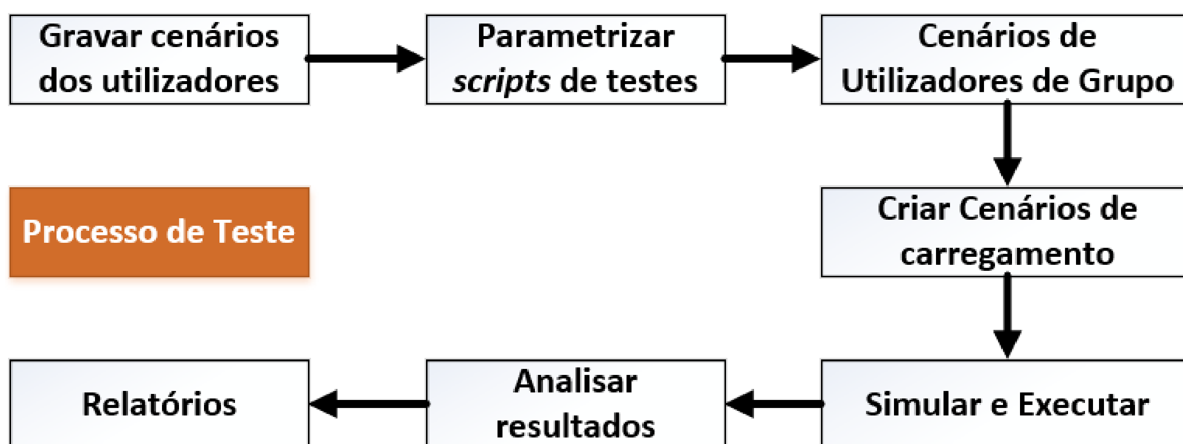


Figura 5 - Testes Aplicacionais Adaptado: (Software Application Testing, 2013)

Os principais objetivos destes testes são (Janssen & Janssen, 2017):

- Compatibilidade e funcionalidade de *hardware* – avalia e garante que se tem o hardware essencial pata efetuar o teste;
- Compatibilidade do sistema operativo – avalia e garante que aplicação seja completamente compatível com diferentes plataformas do sistema operativo;
- Avaliação do código fonte – identifica todos os erros de código e erros na aplicação;
- Usabilidade e funcionalidade – avalia se a aplicação é fácil de utilizar e fornece todas as funcionalidades desejadas;

2.2.1 Paradigmas dos Testes aplicacionais

Existem alguns paradigmas de testes aplicacionais, nesta secção irão ser descritos três tipos de testes: o teste de caixa branca, o teste de caixa preta e o teste de caixa cinzenta, bem como as vantagens e desvantagens de cada um destes testes.

O **teste da caixa branca** baseia-se na estrutura interna da informação do sistema em teste (Arora & Sinha, 2002), ver Figura 6.

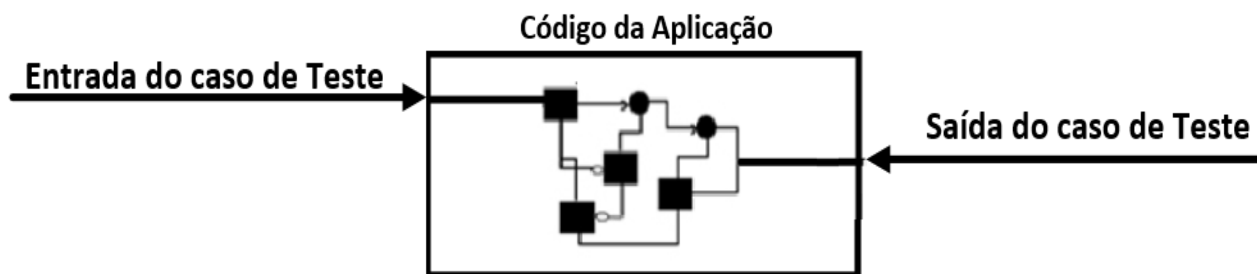


Figura 6 - Teste Caixa Branca Adaptado: (Rongala, 2015)

Este teste é conhecido também como teste estrutural ou teste baseado em código. É uma metodologia que garante e avalia os mecanismos da aplicação, estrutura interna e objetos e os seus componentes. Este método de teste não só averigua o código de acordo com as especificações, mas também visa descobrir as vulnerabilidades da aplicação em estudo (Rongala, 2015).

A estrutura interna da aplicação é testada com as seguintes técnicas:

- *Code Coverage*;
- *Path Coverage;*

Este tipo de paradigma é aplicável á unidade, integração e níveis de sistema de uma fase de teste de *software* (Rongala, 2015).

A realização de um teste caixa branca tem vantagens e desvantagens (Rongala, 2015), observar Tabela 1.

Tabela 1 - Vantagens e Desvantagens dos Teste de Caixa Branca

Vantagens	**Desvantagens**
Otimização de código ao revelar erros escondidos na aplicação em estudo.	Um procedimento complexo e dispendioso que exige a presença de um profissional, especializado em programação e que compreenda a estrutura interna do código
Clareza na estrutura de codificação interna que pode ser útil para mudar o tipo de dados de entrada necessários para um teste da	Necessidade de atualização do *script* de teste após cada modificação.

aplicação.	
Contém todos os caminhos possíveis do código, habilitando uma equipa de testes com capacidade para realizar testes completos.	O teste torna-se ainda mais complicado utilizando o método de teste da caixa branca se a aplicação for grande.
Permite que o programador faça uma introspeção sempre que são feitas novas implementações pela equipa de desenvolvimento.	Algumas condições podem não ser testadas, pois não é revelante testar todo o código.
Os casos de teste serem automatizados.	Os defeitos no código podem não ser detetados.

O **teste de caixa preta** consiste em produzir casos de teste com base nos itens mencionados nas funcionalidades do sistema e onde não se verifica a estrutura e a implementação do código (Arora & Sinha, 2002), ver Figura 7.

Figura 7 - Teste Caixa Preta Adaptado: (Patil, 2017)

A técnica deste paradigma de teste é que não se tem qualquer conhecimento do funcionamento interno da aplicação em causa, só examina os aspetos fundamentais do sistema e não tem a ver com a estrutura lógica do sistema (Khan & Khan, 2012).

A realização de um teste caixa preta pode ter as suas vantagens e desvantagens (Khan & Khan, 2012), observar Tabela 2.

Tabela 2 - Vantagens e Desvantagens do Teste Caixa Preta

Vantagens	**Desvantagens**

Eficiente para o código grande.	Apenas um número selecionado de cenários de teste é realizado.
A perceção do *tester* é muito simples.	Sem especificações claras de se perceber, os casos de teste são difíceis de desenhar.
A perspetiva dos utilizadores está claramente separada da perspetiva de desenvolvedores.	Testes ineficientes.

O **teste da caixa cinzenta** tem como objetivo procurar os defeitos, se houver, devido a uma estrutura inadequada ou uso indevido da aplicação. Este teste também é conhecido como teste translúcido. O teste da caixa cinzenta é apropriado para aplicativos da Web (Acharya & Pandya, 2012), ver Figura 8.

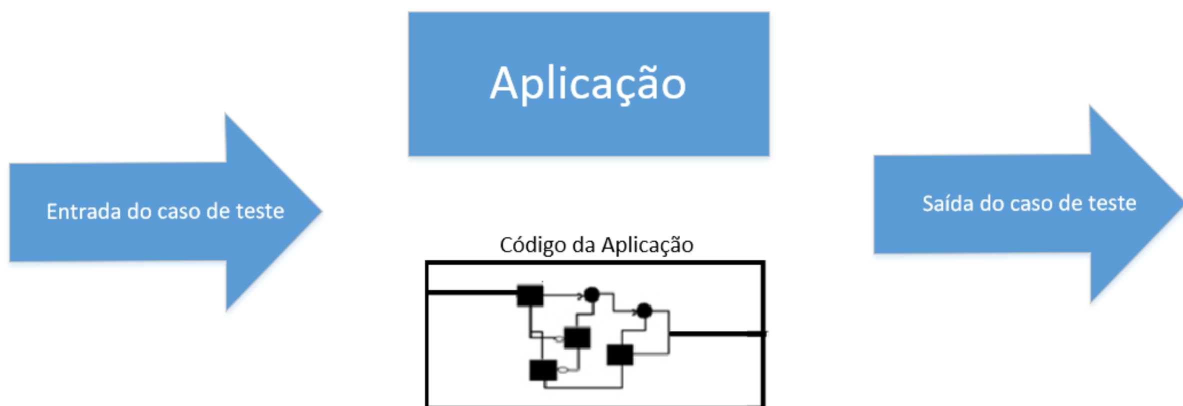


Figura 8 - Teste Caixa Cinzenta Adaptado: (Basic software testing v2.20, 2015)

A realização de um teste caixa cinzenta pode ter as suas vantagens e desvantagens (Acharya & Pandya, 2012), observar Tabela 3.

Tabela 3 - Vantagens e Desvantagens do Teste Caixa Cinzenta

Vantagens	**Desvantagens**
Oferece benefícios combinados.	Uma vez que o código fonte ou os binários não estão disponíveis, a capacidade de

	percorrer caminhos de código ainda é limitada pelos testes deduzidos através da informação disponível.
A caixa cinzenta não depende do acesso ao código, é baseado em definições de interface e especificações funcionais e arquitetura de aplicativos.	Dependente da correta manipulação das exceções no código.
O *tester* pode criar cenários de teste inteligentes, especialmente em relação ao tratamento de dados, protocolos de comunicação e tratamento de exceções.	Dificuldade de identificação de defeitos.

2.3 Testes Funcionais

A principal técnica utilizada nos testes funcionais é a divisão em subconjuntos das entradas de valores, de acordo com as funcionalidades do *software* e selecionando a melhor abordagem a ser utilizada para se obter a validação dos erros e confiabilidade (Silva, Alves, & Andrade, 2015).

Os testes funcionais ocorrem de uma forma mais eficiente e rápida, possibilitando encontrar as não conformidades do *software* em relação aos requisitos do sistema (Leal, 2008).

O teste funcional é um modo de verificar o *software* para garantir que tem todas as funcionalidades necessárias especificadas nos seus requisitos funcionais (Janssen & Janssen, 2017).

Um teste funcional é uma maneira de avaliar o comportamento do *software,* sem que o funcionamento interno do software seja do conhecimento do *tester*, ver Figura 9.

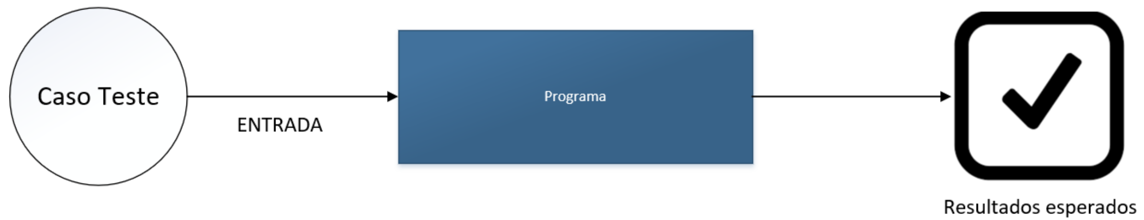


Figura 9 - Teste Funcional Adaptado: (Lopes, 2009)

Os testes funcionais têm a sua estratégia de teste, ou seja, primeiro temos de perceber quais as necessidades, as características e os requisitos do sistema para podermos dizer que se está perante a rastreabilidade, isto é, poder ter a capacidade ou a possibilidade de ser investigado, para depois ser analisado e desenhado, bem como a validação, testes e documentos dos utilizadores, ver Figura 10.

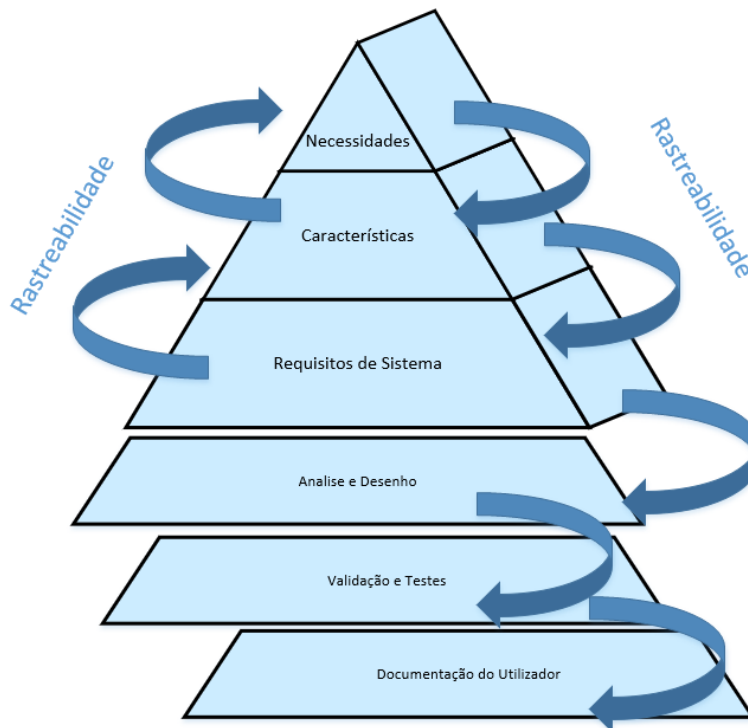


Figura 10 - Estratégia do Teste Funcional Adaptado: (Lopes, 2009)

Os testes funcionais são executados através da concretização de casos de testes e fluxos que utilizam dados válidos e inválidos onde irão averiguar se os resultados apresentados ocorrem conforme o esperado. Deve-se examinar se as mensagens de erro apresentadas são apropriadas quando os dados inválidos são utilizados. Deve-se analisar se a regra de negócio é corretamente aplicada no *software* (Santiago, 2010).

A realização de um teste funcional tem vantagens (Ellauri, s.d.) e desvantagens (Santiago, 2010), observar Tabela 4.

Tabela 4 - Vantagens e Desvantagens dos Testes Funcionais

Vantagens	**Desvantagens**
Melhora a qualidade da aplicação.	Funções incorretas.
Economiza o custo de trabalho.	Funções não implementadas.
Obtém *know-how* e treino para *tester* e programadores.	Erros de Interface.
Apoiar os processos de testes com ferramentas apropriadas.	Erros de desempenho.
Obter testes de qualidade.	Erros de inicialização e finalização.

2.4 Testes unitários

Os testes unitários são uma forma de testar unidades individuais de código fonte, unidades que podem ser métodos, classes, funcionalidades ou módulos. O objetivo deste teste é mostrar que cada unidade reflete corretamente a especificação, como exemplo deste teste ver Figura 11.

```
[Test]
public void TesteDePotenciacao()
{
    // Arrange
    var calculadora = new Calculadora();

    // Act
    var resultado = calculadora.Eleva(3, 4);

    // Assert
    Assert.Equal(81, resultado);
}
```

Figura 11 - Exemplo de um Teste Unitário (Tercete, 2013)

Os testes unitários têm como beneficio:

- Garantir que os problemas são descobertos o quanto antes;
- Facilitar a manutenção do código;
- Servir como documentação;
- Ajudar a manter o código fácil de se entender;

Por exemplo, no ambiente m desenvolvimento do *Visual Studio* é possível identificar o estado dos testes unitários conforme se pode verificar da Figura 12.

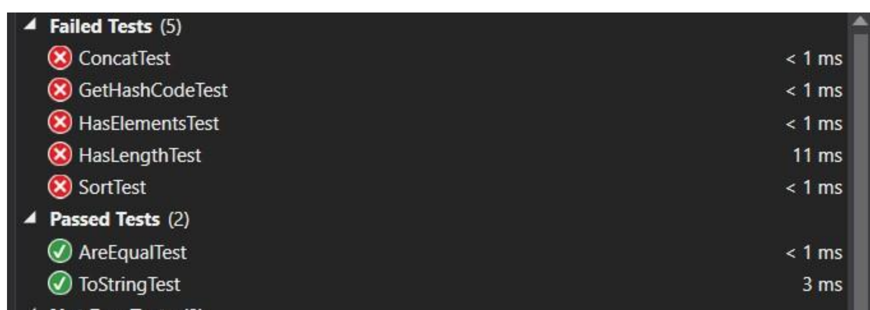


Figura 12 - Testes Unitários

O teste unitário tem como benefício garantir que os problemas são descobertos cedo, facilitar a manutenção do código, servir como documentação e ajudam a melhorar o design do seu código e tornar melhor o desenvolvimento da aplicação (Santos, 2016).

Um teste unitário tem um ciclo com os seguintes passos (Bahia, 2015), ver Figura 13:

- **Avaliação** – determina se o sistema se está a comportar como desejado ou se será necessário fazer novas revisões;
- **Planeamento** – define-se neste passo o que se pretende testar e a forma de como será feito o teste;
- **Preparação** – consiste em preparar o ambiente para se realizarem os testes;
- **Execução** – execução de tudo o que foi planeado e montado;

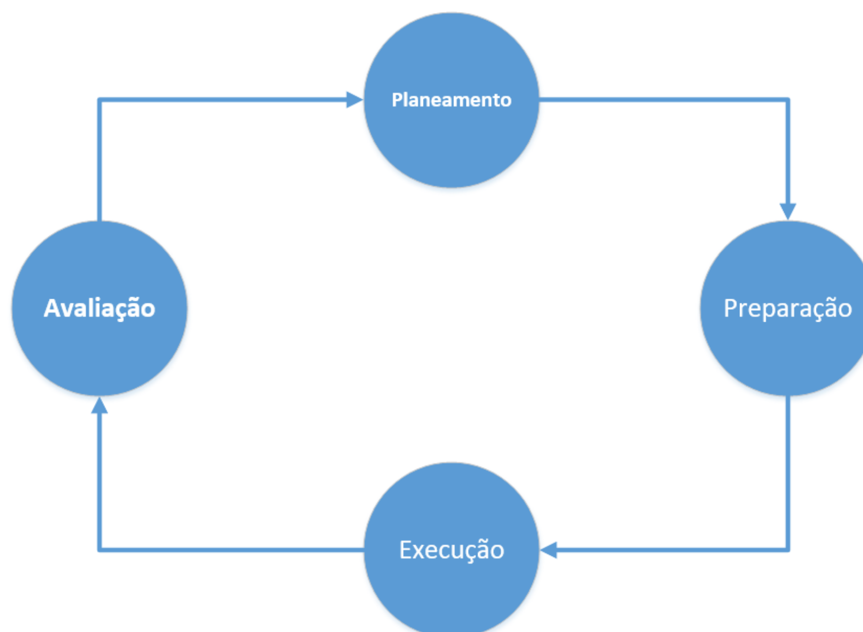


Figura 13 - Ciclo de um teste unitário. Adaptado (Bahia, 2015):

A realização de um teste unitário tem vantagens (Thiago, 2011) e desvantagens (Paulo, 2015), observar Tabela 5.

Tabela 5 - Vantagens e Desvantagens dos Testes Unitários

Vantagens	**Desvantagens**
Maior cobertura de teste.	Complexidade que traz ao *refactoring*.
Previnem regressão.	Difícil repensar os componentes.
Incentivam o *refactoring*.	Manter os testes já desenvolvidos.
Evitam longas sessões de *debug*.	Falta de cultura de testes automatizados de muitos dos programadores.
Servem como documentação.	

2.4.3 *Code Coverage*

Code coverage é uma medida usada para descrever a percentagem de código que tem probabilidade de não ocorrerem *bugs*. Sendo assim, um código com uma elevada percentagem de *code coverage*, tem menor probabilidade de apresentar *bugs* em relação a um bloco de código com menor percentagem. Esta medida é usada em testes unitários e pode ser aferida na Figura 14.

Hierarchy	Not Covered (Blocks)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
argil.utils.dll	7151	98,59%	102	1,41%
{} [illegible]	3278	97,15%	96	2,85%
ArrayUtils	81	59,56%	55	40,44%
AreEqual(System.Array, Sys...	0	0,00%	33	100,00%
Concat(System.Array, Syste...	35	100,00%	0	0,00%
GetHashCode(System.Array)	16	100,00%	0	0,00%
HasElements(System.Colle...	13	100,00%	0	0,00%
HasLength(System.Collecti...	6	100,00%	0	0,00%
Sort(System.Array, System....	11	100,00%	0	0,00%
ToString(System.Array)	0	0,00%	22	100,00%

Figura 14 - Análise do code coverage

O objetivo da realização destes testes é encontrar falhas de funcionamento dentro de uma pequena parte do sistema, funcionando independentemente do todo. Estes testes são feitos pelo próprio programador. Este, geralmente, é mecanizado através de ferramentas como por exemplo *Junit*, *OHPUnix*, *XXXUnit* entre outras. Este tipo de teste precisa estar sempre atualizado e de acordo com as regas de negócio atuais do sistema.

O alvo destes testes são métodos, classes, ou seja, as menores unidades do sistema (Farias, 2017), ver Figura 15.

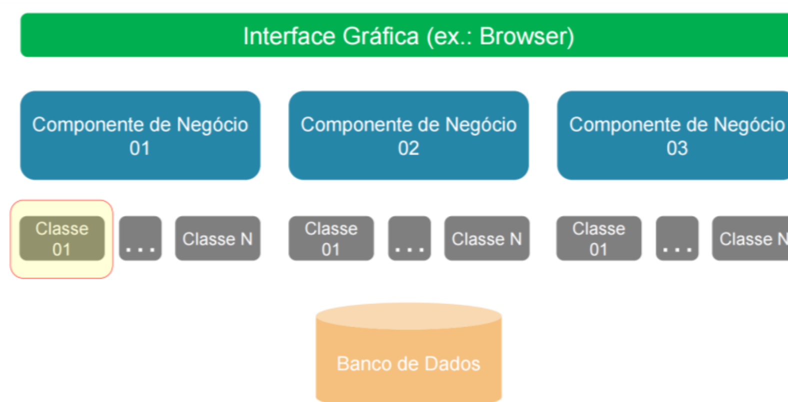


Figura 15 - Teste Unitário (Farias, 2017)

2.5 Testes através da especificação

Um teste de *software* feito através da especificação deve ser abordado de duas maneiras: verificação e clareza do documento e utilização da documentação em conjunto com o programa de *software*. É importante que haja um utilizador que valide a documentação onde todas as discrepâncias devem ser anotadas e, quando existir ambiguidade no documento, a mesma deve ser corrigida. (Ferreira, 2010).

O **teste feito através da especificação** consiste em verificar se o requisito (ver informação mais abaixo) para aquele desenvolvimento está de acordo com o que está descrito na especificação ou, caso não se encontre de acordo, é preciso corrigir o código.

Na **elaboração da especificação**, a linguagem utilizada nos requisitos tem de ser bem descrita para garantir que os programadores interpretam com maior fiabilidade as intenções dos *stakeholders*. A vantagem de utilização da linguagem de especificação dentro da equipa poderá ajudar na melhor compreensão dos requisitos, pois será possível identificar e definir os casos de teste mais facilmente e será provável identificar e corrigir erros desde o início até ao fim do projeto, fazendo com que os custos da aplicação sejam menores (Ferreira, 2010).

As desvantagens dos testes através da especificação são: dificuldade seguir a especificação; dificuldade em testar especificações formais; o cliente poderá não estar familiarizado com a linguagem de especificação.

A Figura 16 demonstra a forma como um projeto de especificação formal se encaixa na estrutura tradicional de desenvolvimento de *software*. Criar uma especificação formal detalhada, implica uma análise especifica do sistema que divulga erros e inconsistências na especificação informal, permitindo assim, eliminar ambiguidades e aumentar a exigência dos requisitos a serem atingidos (Figueiredo, Neve, Magalhães, & Pinto, 2002).

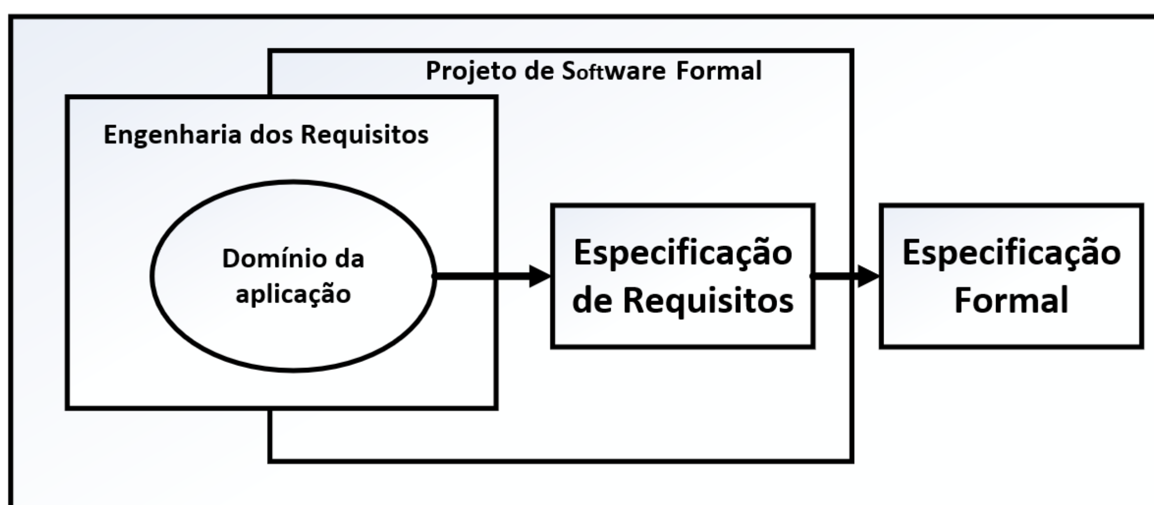


Figura 16 - Especificação de Requisitos e Especificação Foram Adaptado: (Figueiredo, Neve, Magalhães, & Pinto, 2002)

Uma **especificação formal** é escrita numa linguagem onde o vocabulário, sintaxe e semântica se encontram definidos formalmente, tendo a matemática como fundamento para este tipo de especificação formal, então como pode ser utilizada no desenvolvimento de *software*? (Costa & Gonçalves, 2012)

Uma **especificação** é um modo de descrever o *software* e seus requisitos, assim, quando se pretende especificar um novo *software*, são colocados os detalhes dele concomitantemente com os detalhes do mundo real onde se deseja que seja aplicado. A especificação irá servir de contacto entre o cliente e equipa de desenvolvimento, ajudando a registar o que será desenvolvido e como será desenvolvido. Este documento é utilizado em todas as etapas: projeto, codificação e os seus respetivos testes. Este tipo de especificação é a mais utilizada (Costa & Gonçalves, 2012).

O esquema da Figura 17, demonstra os métodos utilizados neste tipo de especiação.

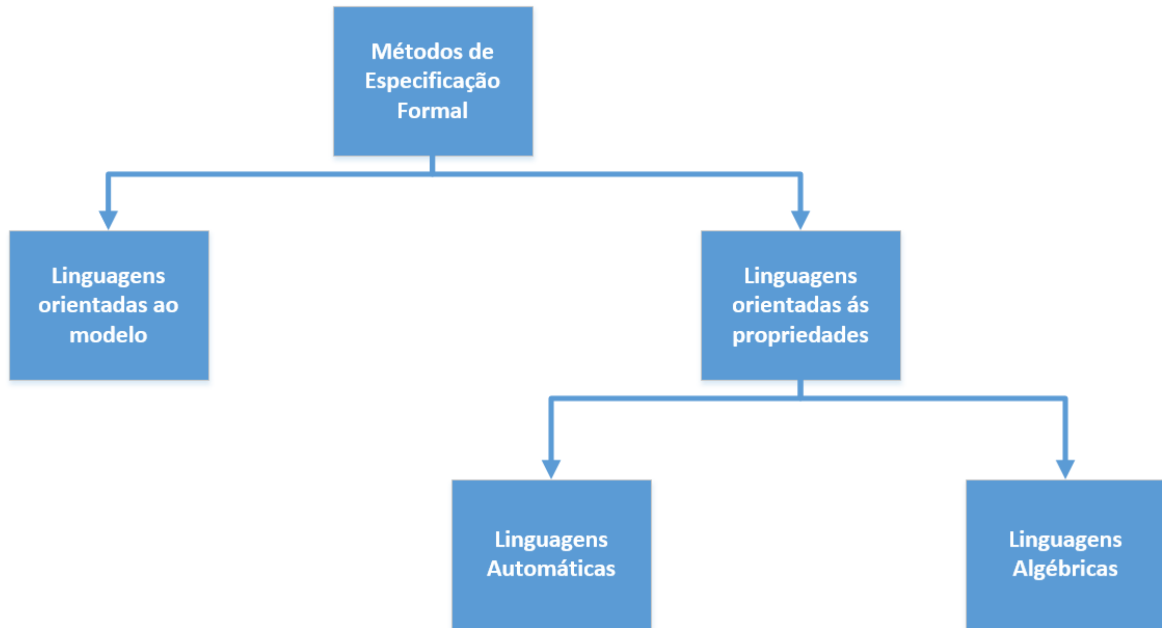


Figura 17 - Especificação Formal – Métodos Adaptado: (Carreira, 2014)

A utilização da especificação formal pode ter as suas vantagens e desvantagens (Figueiredo, Neve, Magalhães, & Pinto, 2002), observar Tabela 6.

Tabela 6 - Vantagens e Desvantagens da Especificação Formal

Vantagens	**Desvantagens**
Compreender melhor os requisitos do sistema.	Existe alguma resistência em adotar novas técnicas quando o lucro não é óbvio.
Compreender a forma de implementação de *software* funcional.	É difícil mostrar que o elevado custo de desenvolvimento da especificação formal, na fase inicial, é baixo.
Refere-se a uma linguagem matemática, onde a especificação é precisa e não ambígua.	Os clientes não se encontram familiarizados com as linguagens deste tipo de especificação.

Utilizada para reconhecer os casos de teste mais importantes para o *software* a desenvolver.	A maioria do trabalho desenvolvido na especificação formal é dirigido ao desenvolvimento de línguas, havendo poucas ferramentas que as implementem.
Processadas automaticamente, através de ferramentas de *software* que aprovam o desenvolvimento, compreensão e *debug*.	
O custo final do *software* pode ser reduzido, dado que estes métodos permitem corrigir erros numa fase inicial, o que se torna mais barato que a correção após a implementação.	

A **especificação de requisitos** tem como objetivo conseguir um *software* de melhor qualidade. que satisfaçam as necessidades dos clientes dentro de prazo e orçamento acordados.

Um requisito (ver mais abaixo informação) pode ser entendido como uma função, restrição ou propriedade que deve ser fornecida para satisfazer as necessidades dos utilizadores do sistema.

Este tipo de especificação descreve um serviço ou uma lentidão (Macoratti, 2012).

Com análise detalhada da Figura 18, pode-se perceber em que consiste a especificação de requisitos.

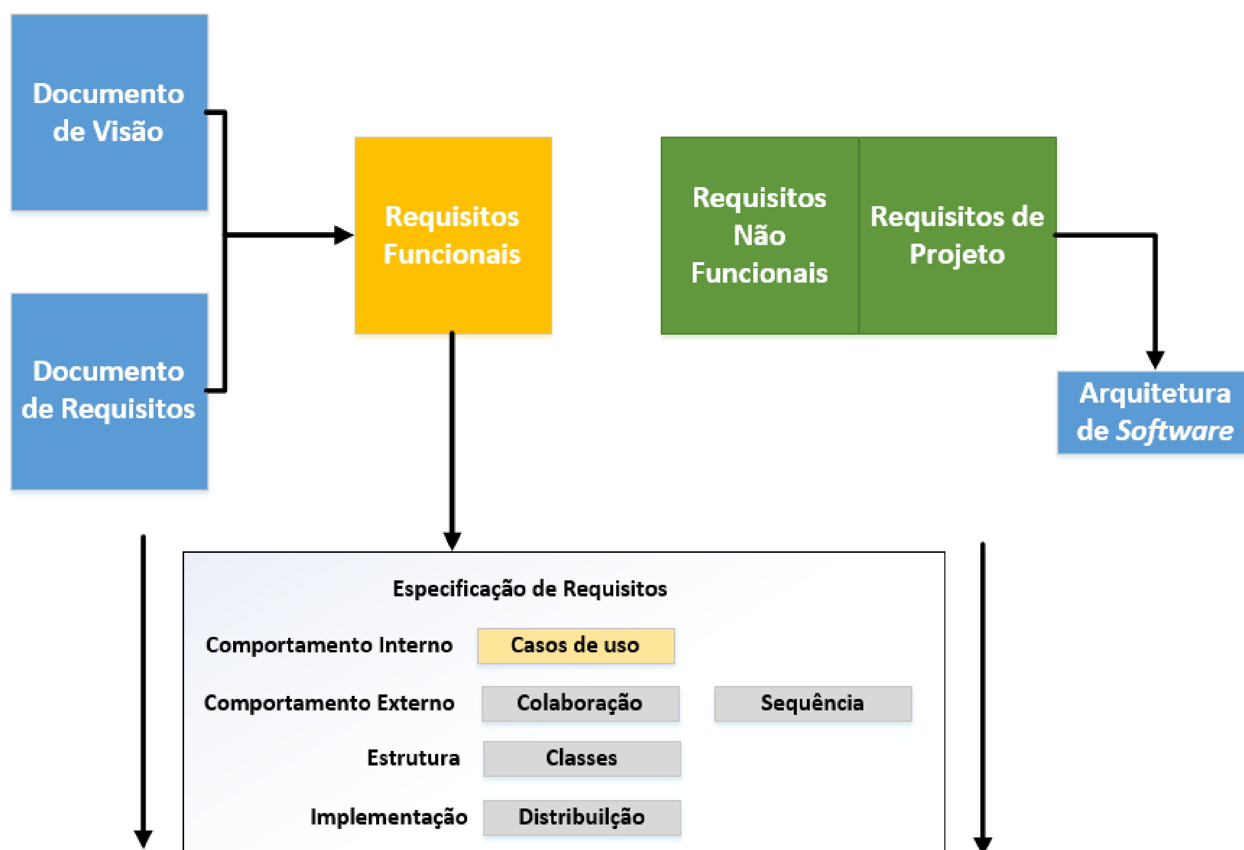


Figura 18 - Especificação Formal Adaptado: (Andrade, 2016)

Requisitos consistem na definição documentada de um determinado comportamento que um produto deve ter. Um conjunto de requisitos é normalmente utilizado para a fase de um projeto onde são especificadas as necessidades consideradas importantes para o desenvolvimento.

Um **requisito** pode ser definido como um estado de um *software* que deve ser implementado por um sistema para obter um determinado fim. Todo o projeto de *software* tem um conjunto de requisitos definidos pelos utilizadores que irão utilizar o *software,* desenvolvido de acordo com o pedido do cliente (Engholm, 2013).

Através da analise da Figura 19 podemos perceber bem qual a definição de um requisito dentro de um projeto.

Figura 19 - Definição de requisito Adaptado: (crvs-dgb, s.d.)

2.5.1 Requisitos Funcionais e Não Funcionais

Os requisitos estão divididos em dois: os requisitos funcionais e os requisitos não funcionais.

Um **requisito funcional** define uma função de um sistema de *software*, pois este representa o que o *software* faz em termos de tarefas. Estes requisitos podem ser, por exemplo, cálculos, detalhes técnicos e manipulação de dados.

Quando se fala de um requisito funcional está-se a referir a uma requisição de uma função que um *software* deverá realizar, ou seja, a solicitação que um *software* deverá concretizar (Ventura , 2016).

Uma funcionalidade pode concretizar vários requisitos funcionais, ou seja, significa que numa funcionalidade, um ou mais requisitos funcionais podem ser considerados, não necessariamente apenas um. Se pensarmos em multiplicidade, uma funcionalidade pode realizar um ou muitos requisitos funcionais (1.. *), ver Figura 20 (Ventura , 2016).

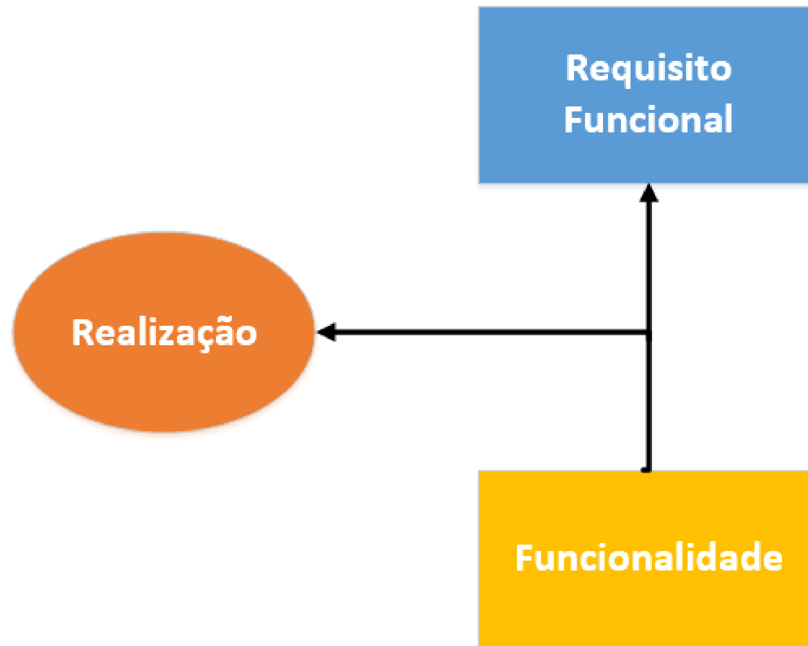


Figura 20 - Requisito Funcional Adaptado: (Ventura , 2016)

O plano de implementação de requisitos funcionais é detalhado no projeto do sistema enquanto que nos requisitos não-funcionais o plano de implementação é detalhado na arquitetura do sistema.

Um **requisito não funcional** está relacionado com o uso da aplicação em termos de desempenho, usabilidade, segurança, disponibilidade, manutenção e as tecnologias que estão envolvidas.

Este tipo de requisito tem como objetivo atender a requisitos do sistema que não são requisitos funcionais, mas que constroem parte do escopo do sistema. Este tipo de requisito pode ou não estar associado a um requisito funcional, onde esta associação é muito comum em requisitos não funcionais de integração de sistemas (Ventura , 2016).

2.5.2. Técnicas para levantamento de requisitos

O início para o desenvolvimento de software é o levantamento de requisitos, sendo esta atividade repetida em todas as demais etapas de desenvolvimento de software. O levantamento de requisitos tem as seguintes atividades (Bedani , 2017):

- **Compreensão do domínio** – onde os analistas desenvolvem a sua compreensão do domínio da aplicação;

- **Recolha de requisitos -** processo de interação com os *stakeholders* do sistema para descobrir requisitos;

- **Classificação -** atividade que se considera o conjunto não estruturado dos requisitos e os organiza em grupos coerentes;

- **Resolução de conflitos -** múltiplos *stakeholders* estão envolvidos, os requisitos mostrarão conflitos;

- **Definição das prioridades** - qualquer conjunto de requisitos, alguns serão mais importantes do que outros, onde esse estágio envolve interação com os *stakeholders* para a definição dos requisitos mais importantes (Bedani , 2017);

- **Verificação de requisitos –** requisitos que são verificados para descobrir se estão completos e consistentes;

O levantamento e análise de requisitos é um processo iterativo, com uma contínua aprovação de uma atividade para outra (Bedani , 2017), conforme ilustrado pela Figura 21.

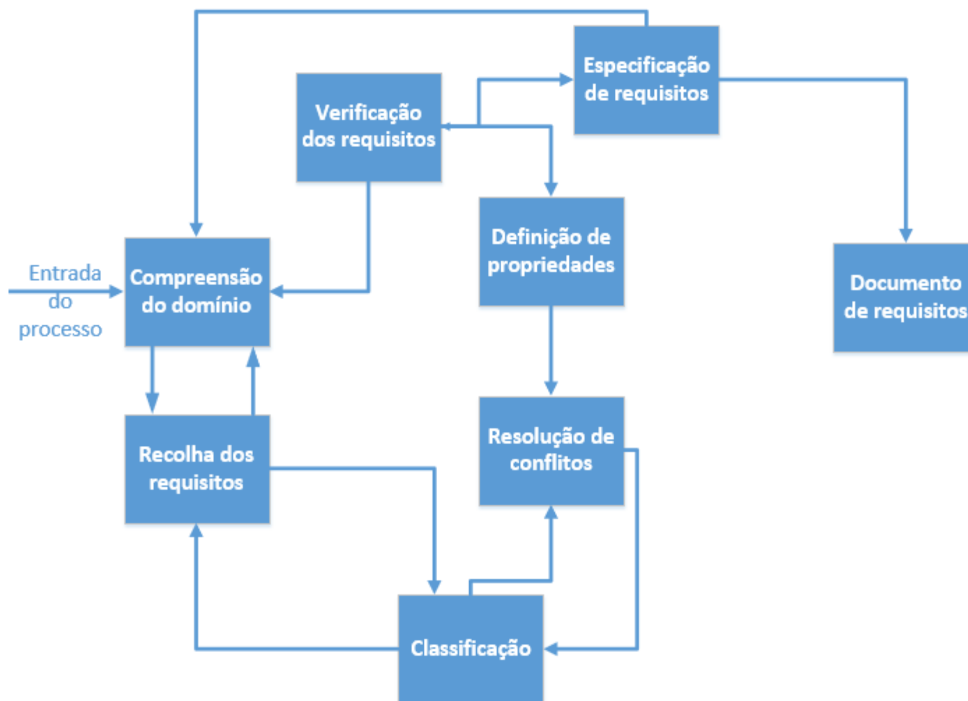


Figura 21 - Levantamento e analise de requisitos Adaptado: (Benadi, 2017)

2.6 Importância da documentação

A elaboração de documentação é muito importante nos projetos porque na documentação constam os requisitos fundamentais na elaboração do *softwar*e e o papel do *tester* da equipa é seguir requisito a requisito e, ao efetuar o teste, ver se foram ou não cumpridos.

Espinha, evidencia a importância da documentação da seguinte forma:

> *"Preservar um registo de todas as fases do projeto, de como cada item foi realizado, que decisões foram tomadas, como e porquê, pode parecer preciosismo, mas a verdade é que a documentação em projetos é uma forma de proteger a equipa em relação a todo o desenvolvimento do trabalho. Mesmo sendo tão importante, é comum ver-se projetos com pouca documentação, o que pode ter um impacto negativo na sua finalização, seja pela falta de um registo para comparar o previsto e o realizado ou pela falta de credibilidade repassada para o cliente."* (Espinha, 2016)

Existem alguns passos para existir uma boa documentação na equipa que fazem com que a gestão do desenvolvimento seja mais fácil que são (Espinha, 2016):

- **Comunicação clara e acertada deve estar registada -** comunicação entre a equipa de desenvolvimento deve ser clara e eficiente, não deixando aberturas para dúvidas ou interpretações diferenciadas;

- **Histórico de ações para uma avaliação mais minuciosa** - conservar um histórico sobre o projeto também é um dos fatores que fazem com que a documentação seja tão importante;

- **Maior controle para o desenvolvimento do projeto -** ambiente de projetos é excessivamente dinâmico e incerto, o que coloca, muitas vezes, a equipa frente a situações que podem fugir ao controle;

- **Alinhamento de informações na inserção de novos membros na equipa -** importância da documentação em projetos também está em preservar um registo fiel do desenrolar das atividades para que, na inserção de um novo membro, este se possa inteirar de todos os acontecimentos e assim colaborar com maior eficácia nas atividades**;**

- **Base para tomada de decisões -** tomada de decisão no projeto não pode ser aleatória, deve estar baseada em fatos e dados prováveis, os quais podem e devem estar assinalados na documentação do projeto;

- **Reporte de *status* para os *stakeholders* -** outra funcionalidade e importância da documentação em projetos é manter todos os *stakeholders* esclarecidos sobre o desenvolvimento, com dados definidos e frequentes que mantêm todos alinhados e com as mesmas expectativas. Ter um registo permite que a equipa de desenvolvimento se mantenha constantemente destacada nas melhores práticas e consiga otimizar tempo e recursos a cada nova atividade, tornando-se mais produtiva e eficaz cada vez mais.

As falhas apresentadas durante o processo de teste de *software* devem ser registadas com informações suficientes para que este defeito possa ser reproduzido, analisado e corrigido (IEEE, 1998).

3. Estudo de caso

É neste capítulo que irá ser realizado o estudo do caso, isto é, o ponto de partida para a realização do estudo de modo a obter a informação sobre os testes de *software*.

Na *Estamos Juntos* a metodologia de trabalho passa pela criação e acompanhamento de tarefas designadas por *tickets*, estas tarefas, em regra geral, são criadas pelos *stakeholders* do projeto e apresentam melhorias ou novas funcionalidades a acrescentar ou remover do projeto. Os *tickets* são usualmente atribuídos a uma pessoa que os resolve e reportados ao *tester*, este elemento faz a sua validação e comunicação dos resultados com os *stakeholders*.

3.1 Tickets

Um *ticket* é algo que foi reportado como um erro que acontece na aplicação de qualidade (*Beta*) no decorrer dos testes, podem também existir *tickets* com novas funcionalidades que irão passar a constar na aplicação de qualidade e na de produção.

Normalmente, quando são novas funcionalidades, cabe ao *tester* ler a especificação sobre essa nova função, reunindo, de imediato, com os programadores responsáveis por esses desenvolvimentos e depois criar os respetivos *ticket*s

O foco principal dos programadores é fazer o código que garante que as funcionalidades descritas nos *tickets* são implementadas, enquanto que os *testers*, fazem a validação e documentação do que foi feito pelos programadores e comunicam aos *stakeholders*, e estes, caso entendam, podem também fazer os seus testes no âmbito do *ticket*, como podemos ver na Figura 22.

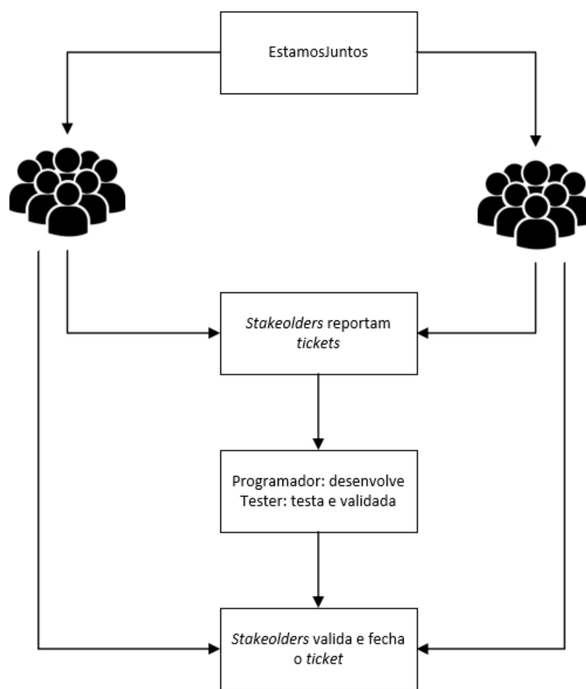


Figura 22 - Histórico dos Tickets

3.1.1 Gestão de tickets

A gestão dos *tickets* da *Estamos Juntos* é feita na ferramenta *Visual Studio Team Services*, ver Figura 23.

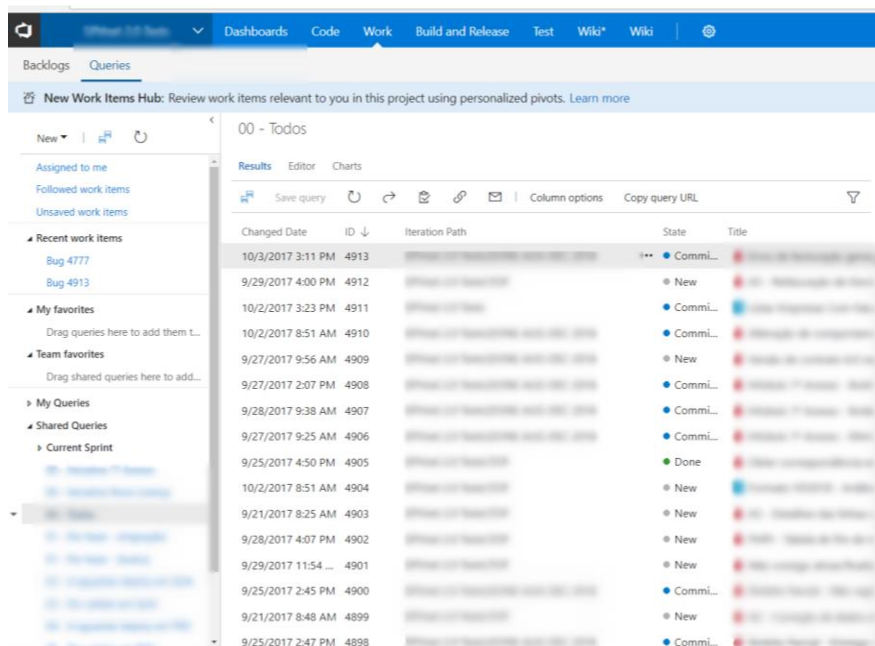


Figura 23 - Gestão de tickets

Esta ferramenta possibilita aceder a todos os *tickets*, permite definir prioridades, criar novos *tickets* com novos desenvolvimentos, ou *tickets* que reportam erros, atribuir *tickets* a membros da equipa, definir os estados do *ticket*, validar / rejeitar o *ticket* após concluídos os desenvolvimentos, entre outras funcionalidades.

Esta ferramenta é muito fácil de utilizar por todos elementos da equipa e fácil de perceber qual o elemento ou elementos que pegaram em determinados *tickets* e saber quem foi a pessoa responsável pela sua criação, neste caso o cliente.

3.1.2 Estados dos tickets

Na Figura 24 pode observar-se pelo esquema em que estados um *ticket* se pode encontrar.

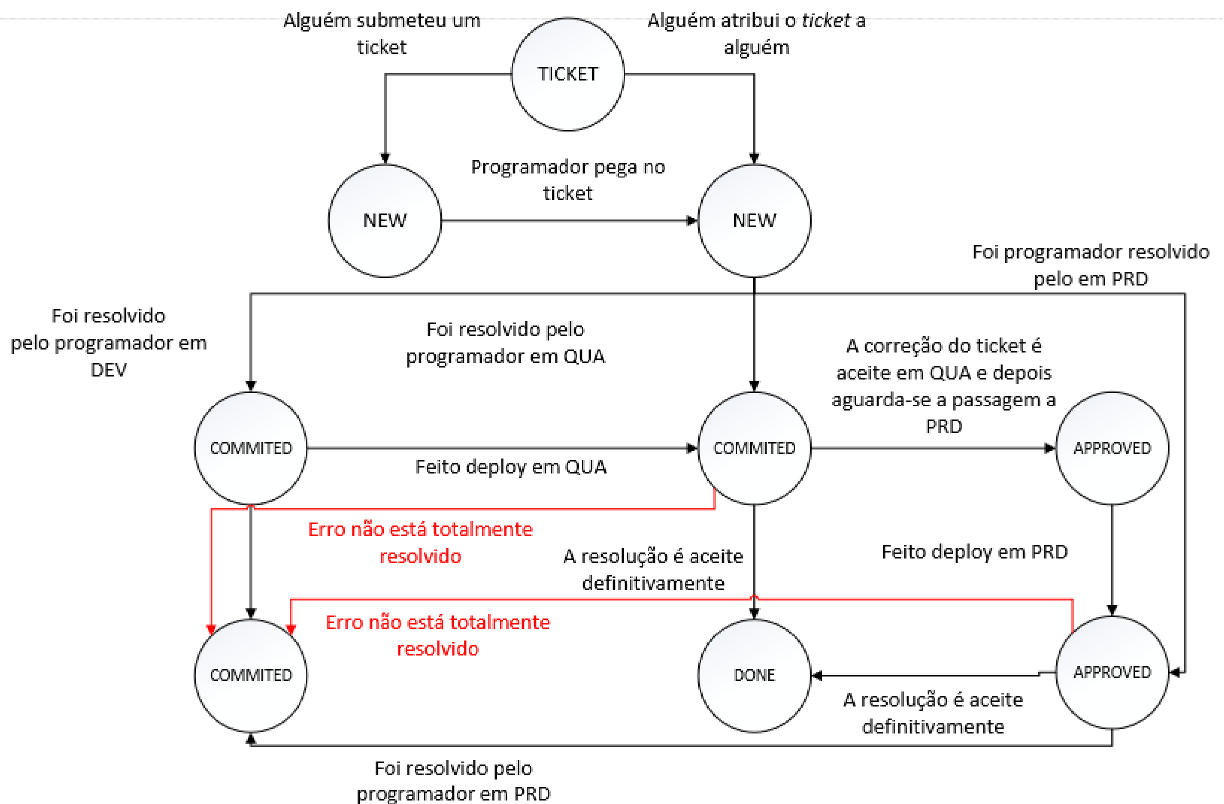


Figura 24 - Estado dos tickets

Os *tickets* que são criados devem conter o estado de *New* e serão atribuídos ao utilizador genérico da empresa. Posteriormente, quando algum elemento da equipa de programação opta por fazer a sua resolução, então esse *ticket* fica-lhe atribuído.

Uma vez que se encontra resolvido e supondo que a correção feita não afeta diretamente o

ambiente de QUA ou PRD, então muda o estado para *Committed* e o *ticket* é atribuído ao utilizador genérico do *deploy*. Se, por sua vez, a resolução feita afeta diretamente no ambiente, então deve-se atribuir à pessoa que reportou o *ticket* e se, eventualmente, teve de fazer a resolução no ambiente de QUA deve ficar com estado *Committed*, se foi em PRD que a resolução do *ticket* foi feita, então passa a ter o estado como *Approved*.

Uma vez que correção já se encontra no ambiente de QUA, então o *ticket* será atribuído à pessoa que o irá validar e, ao fazer o teste, poderá acontecer uma das seguintes situações:

- A pessoa valida e muda o estado para *Approved* e atribuiu ao utilizador genérico;
- A pessoa não valida e, nesse caso, deve retornar o *ticket* a quem fez o trabalho;

Após este passo estando com a correção já no ambiente de PRD, irá ocorrer exatamente o mesmo:

- A pessoa valida e muda o estado para *Done*. É irrelevante a quem atribui, mas recomenda-se tirar a atribuição, para não ficar associado a ninguém;
- A pessoa não valida e nesse caso deverá retorná-lo a quem fez o trabalho, ou seja, o programador responsável;

Quando se retorna *tickets*, só se irá mudar o estado no caso em que seja de PRD e só aí se encontraram alguns erros. Então, nessa situação, o estado passa de *Approved* para *Committed*, sendo atribuído à pessoa que inicialmente estava a trabalhar no seu desenvolvimento.

Realce-se que o significado de Approved não quer dizer que foi aprovado por alguém, ou seja, significa apenas que o trabalho já está em PRD.

De igual forma, atribuir um *ticket* ao utilizador genérico significa sempre que este está na fila a aguardar o respetivo *deploymen*t no próximo ambiente em QUA, se o estado do *ticket* for Committed ou em PRD se o estado do *ticket* for Approved.

Para a realização do estudo, foi feita uma análise em detalhe de como a equipa desenvolve e faz a gestão dos tickets semana após semana. Em cada semana, existe um novo conjunto de *tickets* com novas prioridades a serem cumpridas, onde podem ser mudadas durante o

decorrer da semana se, eventualmente, surgir alguma funcionalidade mais importante e que tenha um prazo mais apertado.

Portanto, param-se as outras funcionalidades e começa-se na que surgiu. O mesmo acontece quando é reportado um *ticket* urgente que, na maioria das vezes, acontece no ambiente de produção e, por isso, convém ficar corrigido o mais rapidamente possível. Sempre que surge este tipo de *ticket,* suspende-se o que se está a fazer e resolve-se o problema surgido e, após a sua correção, é aplicado uma correção a (*hotfix* como se verá mais baixo) em produção e esse erro, que tinha sido reportado, já não existe em PRD.

Um ticket que é reportado com o estado e a prioridade urgente, significa que o erro se encontra a ocorrer no ambiente de PRD e é uma situação muito preocupante perante o cliente.

Significa que, com as passagens que existiram entre os ambientes QUA e PRD, algumas funcionalidades dos desenvolvimentos dos *tickets* foram quebradas, ver Figura 25.

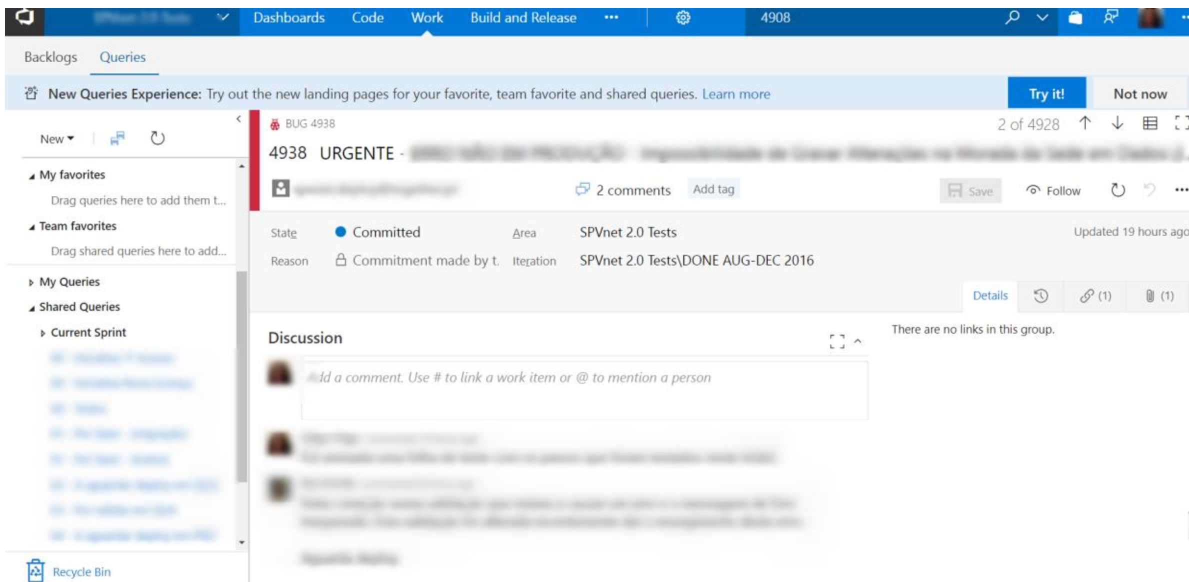


Figura 25 - Ticket Urgente

3.1.4 Prioridades dos Tickets

Na empresa onde a mestranda se encontra a realizar este estudo, os *tickets* são classificados por grau de prioridades e consoante esse grau, os programadores resolvem os *tickets*. A ordem de prioridade é a seguinte:

- Urgente - significa que se devem parar os desenvolvimentos associados ao *ticket* que o programador se encontra a realizar de momento e pegar neste;
- Crítica – este *ticket* classificado assim tem impreterivelmente estar realizado;
- Alta – este *ticket* classificado assim tem que estar desenvolvido, é muito negativo caso isto ainda não tenha sido possível, pois o cliente poderá não aceitar a solução apresentada;
- Média - este *ticket* classificado assim indica que era bom estar resolvido, mas se não estiver é apenas um inconveniente, não sendo impeditivo de ir para PRD;
- Baixa - este *ticket* classificado assim é para ser feito, em momento oportuno, pois não oferece ainda nenhum tipo de problema.

Através da análise durante um intervalo de tempo de registo, ver gráfico da - *Grau de Prioridade dos Tickets*Figura 26, pode-se concluir que a empresa, normalmente, apresenta um grau de prioridade crítica muito elevado, imposto semanalmente pelo cliente, isto é, todos os *tickets* que forem criados semana após semana têm que, ser desenvolvidos rapidamente pela equipa desenvolvimento. Pode-se observar também que os graus médio e baixo têm poucos *tickets*, isto é, não existe muita afluência na criação de novos *tickets* com esse estado. Portanto, conclui-se que a situação critica apresenta um número muito significativo, seguido de Alta e Urgente. Estas são as prioridades mais frequentes definidas na empresa, semana após semana.

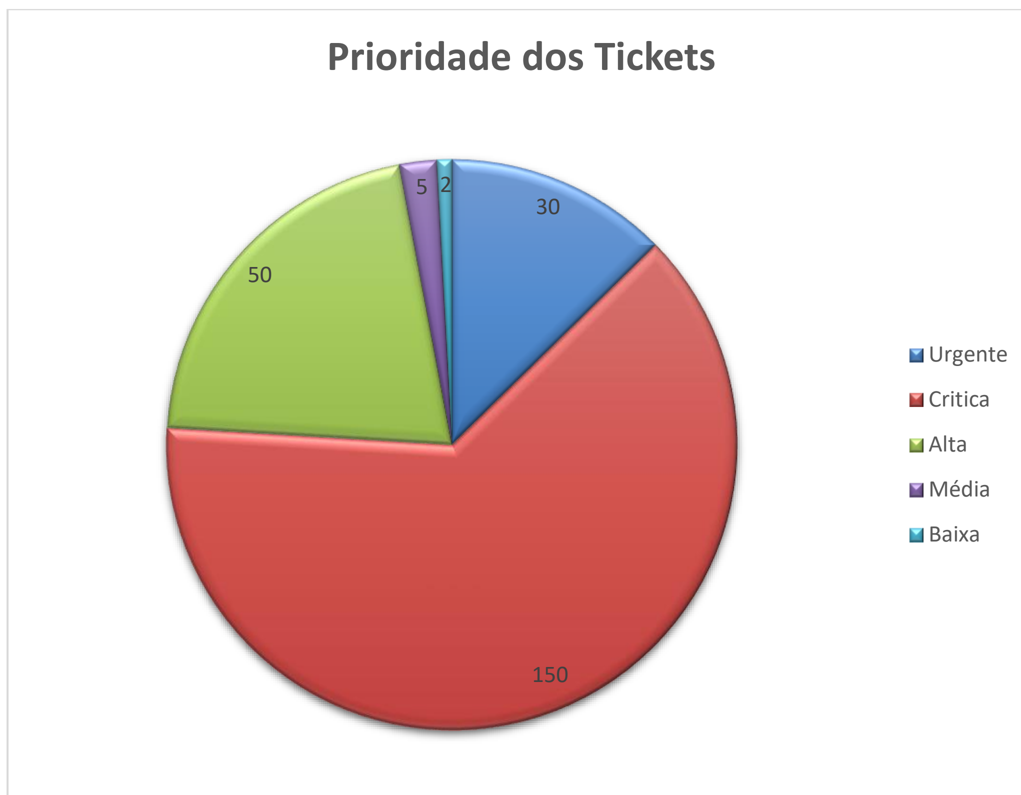


Figura 26 - Grau de Prioridade dos Tickets

Um ***hotfix*** é um código que, por vezes, é chamado de *patch*. Este corrige um erro em PRD. As correções às vezes são empacotadas como um conjunto de correções, chamado de hotfix combinado ou *service pack* (Rouse, 2017).

Esta situação é um pouco rara de acontecer na empresa onde a mestranda se encontra a realizar o estudo, mas quando existem situações urgentes no ambiente de produção e se for para se corrigir o quanto antes, então disponibiliza-se um elemento da equipa e, após a resolução do erro, aplica-se então o *hotfix* em PRD.

Um *hotfix* é um pacote que contêm apenas os desenvolvimentos do erro que for reportado no ticket e posteriormente é aplicado em PRD a fim de corrigir essa situação.

3.2 Documentação dentro da equipa

Dentro da empresa onde a mestranda se encontra a realizar o estudo, nota-se que existe documentação para tudo, uma delas, a mais importante de todas, é o manual da aplicação, isto é, neste manual está documentada toda a informação importante para equipa de

desenvolvimento. Assim, caso algum elemento não se recorde de alguma funcionalidade da aplicação ou caso entre um elemento novo para a equipa, é fundamental a leitura dessa documentação.

Na empresa existem muitos registos documentados através das especificações que vão sendo feitas e, para o registo de *tickets* desenvolvidos, existe uma grelha gerada no Excel onde são registadas as tarefas, ver Tabela 7.

Tabela 7 - Gestão de Tarefas

ID	Módulo	Ticket	Token	Tarefa	Prioridade	Estado	A.	R.	Prazo	R.Prazo	Data Fim	PTA (h)	ETA (h)
1151		4845	4845/1151		lta	Testado	QUA		2017-07-07		2017-07-06	6	0
1152		4846	4846/1152		rítica	Fechado			2017-07-07				
1153		4847	4847/1153		rítica	Testado	QUA		2017-07-07		2017-07-06	1	0
1154		4848	4848/1154		rítica	Testado	QUA		2017-07-07		2017-07-06	2	0
1155		4849	4849/1155		rítica	Em Progress			2017-07-07			12	6
1156		4850	4850/1156		rítica	Em Progress			2017-07-28			8	8
1157		4851	4851/1157		rítica	Testado	PRD		2017-07-07		2017-07-07	1	0
1158		4828	4828/1158		RGENTE	Aberto			2017-07-28			8	8
1159		4853	4853/1159		lta	Fechado	QUA		2017-07-21		2017-07-19	8	0
1160		4854	4854/1160		lta	Aberto			2017-07-28			8	8
1161		4855	4855/1161		rítica	Por Testar	DEV		2017-07-21		2017-07-20	16	8
1162		4852	4852/1162		lta	Em Progress			2017-07-28			8	8
1163		4856	4856/1163		lta	Aberto			2017-07-28			8	8
1164		4857	4857/1164		lta	Aberto			2017-07-28			8	8
1165		4858	4858/1165		rítica	Por Testar	DEV		2017-07-28		2017-07-25	8	0
1166		4859	4859/1166		rítica	Por Testar	DEV		2017-07-28		2017-07-26	8	0
1167		4860	4860/1167		rítica	Aberto			2017-07-28			8	8
1168		4861	4861/1168		rítica	Aberto			2017-07-28			8	8
1169		4862	4862/1169		Crítica	Aberto			2017-07-28			8	8

Podemos observar o esquema para se saber como funciona a coluna dos estados e dos respetivos ambientes através da Figura 27. Existem muitas colunas, mas as mais importantes para a gestão das tarefas serão a coluna do número da *task*, do número do *ticket, do módulo* a que corresponde na aplicação, da descrição da tarefa, o que é preciso fazer ou corrigir, da pessoa responsável, quem irá pegar no *ticket*, do estado que, neste caso existem quatro, ou seja, aberto, por testar, testado e fechado, também contêm uma coluna em que o ambiente se encontra DEV, QUA e PRD e os respetivos prazos e datas.

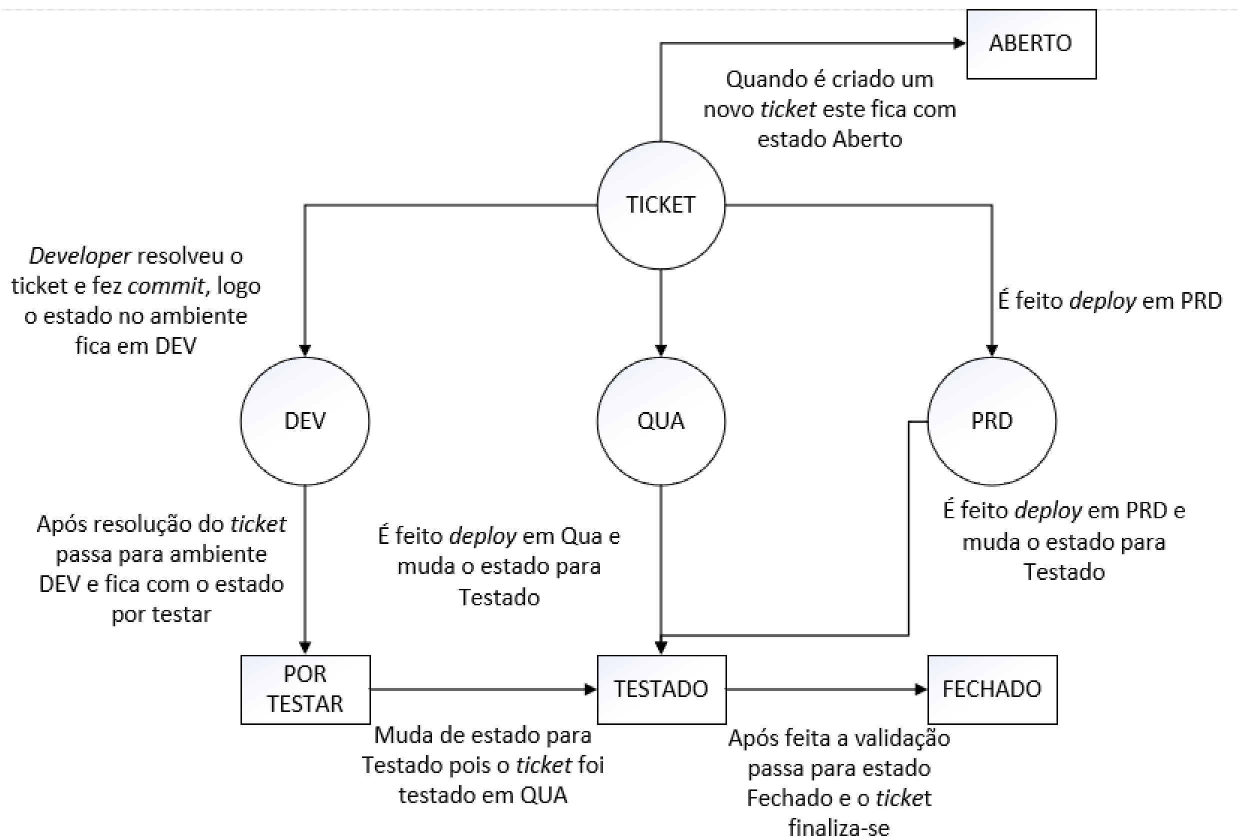


Figura 27 - Gestão dos Estados e Ambientes

Dentro da equipa de desenvolvimento, existe um elemento que todas as semanas faz o levantamento de todos os tickets para saber quantos se encontram fechados, essa informação tem os seguintes passos:

1- Abrir o VST e ir ao separador "*All done*", ver Figura 28. Nesta parte aparecem todos os tickets que se encontram feitos, então agora é ir ao primeiro ticket fazer "CTRL+A" e depois "CRTL+C".

Figura 28 - Tickets All Done

2- Seguidamente o próximo passo é fazer o copiar da tabela que aparece, ver Figura 29.

Figura 29 - Copiar dados da tabela

3- Ir à tabela da folha com a gestão dos tickets fechados, ver Tabela 8, apagar as linhas

referentes aos tickets e copiar a informação da tabela do VST.

Tabela 8 - Fecho de tickets

A	B	C	D
Task List Ticket		ID	Deleted?
3665	NO		
4491	NO		
4544	NO		
3537	NO		
4568	NO		
4569	NO		
4209	NO		
2767	NO		
4267	NO		
2775	NO		
4356	NO		
3242	NO		
4262	NO		
3334	NO		
4389	NO		
4393	NO		
4394	NO		
4486	NO		
3044	NO		
3537	NO		
4569	NO		
3665	NO		
3665	NO		
4569	NO		
4618	NO		
4620	NO		
4625	NO		
3665	NO		
3665	NO		
3665	NO		
3665	NO		
3665	NO		
3665	NO		
3665	NO		
4638	NO		
3665	NO		
3665	NO		
3547	NO		
4089	NO		
4650	NO		
3665	NO		

4- Ir a tabela, que se encontra na gestão de tarefas, e selecionar o filtro referente ao fecho dos *tickets*, ver Tabela 9, copiar a coluna número três, que diz respeito aos *tickets*, para a primeira coluna da folha de fecho de *tickets* e, na segunda coluna, se o estado mudar para "YES" é porque esse *ticket* já se encontra fechado, se o estado for "NO" o *ticket* ainda não está fechado.

Tabela 9 - Filtro fecho de tickets

1200	[illegible]	4841	Ticket #4841 - Task 1200 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-09-01	[illegible]
1203	[illegible]	4875	Ticket #4875 - Task 1203 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-09-01	[illegible]
1204	[illegible]	4888	Ticket #4888 - Task 1204 -	[illegible]	Critica	Testado	PRD	[illegible]	2017-09-08	[illegible]
1207	[illegible]	4886	Ticket #4886 - Task 1207 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-09-01	[illegible]
1210	[illegible]	4890	Ticket #4890 - Task 1210 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-09-08	[illegible]
1214	[illegible]	4894	Ticket #4894 - Task 1214 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-09-08	[illegible]
1217	[illegible]	4895	Ticket #4895 - Task 1217 -	[illegible]	URGENTE	Testado	QUA	[illegible]	2017-09-08	[illegible]
1220	[illegible]	4896	Ticket #4896 - Task 1220 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-09-15	[illegible]
1221	[illegible]	4897	Ticket #4897 - Task 1221 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-09-15	[illegible]
1222	[illegible]	4850	Ticket #4850 - Task 1222 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-09-29	[illegible]
1223	[illegible]	4898	Ticket #4898 - Task 1223 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-09-15	[illegible]
1224	[illegible]	4788	Ticket #4788 - Task 1224 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-09-15	[illegible]
1231	[illegible]	4901	Ticket #4901 - Task 1231 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-09-29	[illegible]
1232	[illegible]	4777	Ticket #4777 - Task 1232 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-10-06	[illegible]
1234	[illegible]	4886	Ticket #4886 - Task 1234 -	[illegible]	Média	Testado	QUA	[illegible]	2017-09-22	[illegible]
1235	[illegible]	4625	Ticket #4625 - Task 1235 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-09-22	[illegible]
1236	[illegible]	4618	Ticket #4618 - Task 1236 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-10-13	[illegible]
1237	[illegible]	4866	Ticket #4866 - Task 1237 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-10-13	[illegible]
1238	[illegible]	4875	Ticket #4875 - Task 1238 -	[illegible]	Critica	Aberto		[illegible]	2017-10-20	[illegible]
1239	[illegible]	4903	Ticket #4903 - Task 1239 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-10-06	[illegible]
1242	[illegible]	4906	Ticket #4906 - Task 1242 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-09-29	[illegible]
1243	[illegible]	4907	Ticket #4907 - Task 1243 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-09-29	[illegible]
1244	[illegible]	4908	Ticket #4908 - Task 1244 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-09-29	[illegible]
1245	[illegible]	4909	Ticket #4909 - Task 1245 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-10-20	[illegible]
1246	[illegible]	4910	Ticket #4910 - Task 1246 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-09-29	[illegible]
1248	[illegible]	4569	Ticket #4569 - Task 1248 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-10-06	[illegible]
1250	[illegible]	4913	Ticket #4913 - Task 1250 -	[illegible]	Critica	Testado	PRD	[illegible]	2017-10-06	[illegible]
1253	[illegible]	4915	Ticket #4915 - Task 1253 -	[illegible]	Alta	Testado	PRD	[illegible]	2017-10-06	[illegible]
1254	[illegible]	4898	Ticket #4898 - Task 1254 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-10-06	[illegible]
1264	[illegible]	4926	Ticket #4926 - Task 1264 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-10-13	[illegible]
1266	[illegible]	4928	Ticket #4928 - Task 1266 -	[illegible]	Critica	Testado	QUA	[illegible]	2017-10-20	[illegible]
1270	[illegible]	4788	Ticket #4788 - Task 1270 -	[illegible]	Média	Testado	QUA	[illegible]	2017-10-13	[illegible]
1271	[illegible]	4931	Ticket #4931 - Task 1271 -	[illegible]	Alta	Testado	QUA	[illegible]	2017-10-20	[illegible]

Um dos documentos importantes dento do seio da empresa é a especificação pois é onde se encontra o registo de como é que o cliente pretende uma determinada funcionalidade ou quando existe alguma nova funcionalidade que tenha de ser desenvolvida. É nela que irão constar os requisitos com a informação pretendida, ver Figura 30.

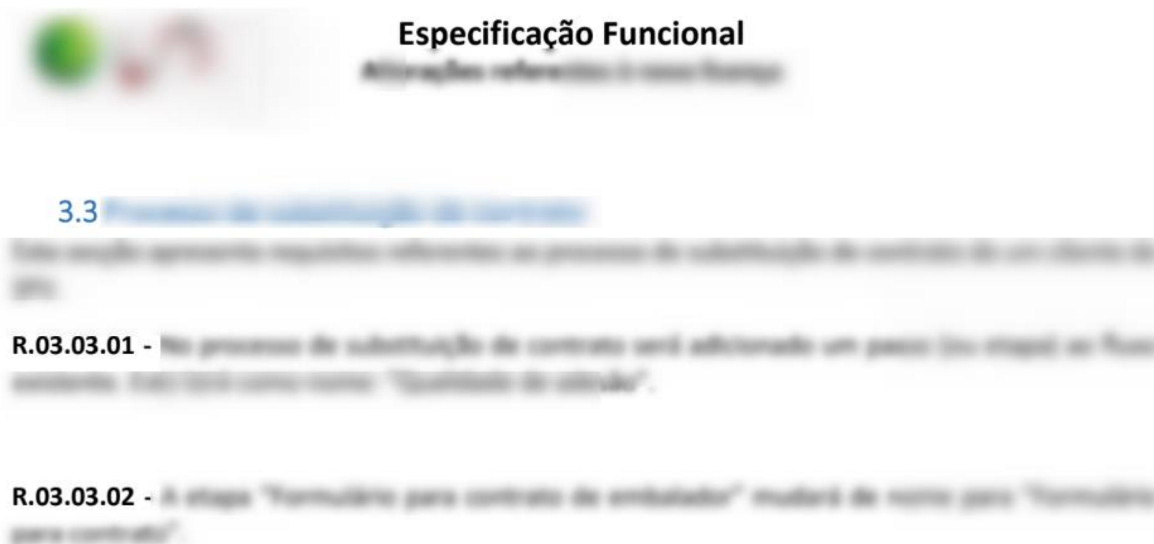


Figura 30 - Constituição de uma especificação

Na empresa onde a mestranda se encontra a realizar o estudo, constatou que a escrita de especificações vai tendo versões novas, ou seja, sempre que surge algo novo, vai-se acrescentado, por exemplo, alguma situação que não se tinha ainda conversado com o cliente ou porque se constatou que era preciso retificar um requisito, ou por ter havido necessidade de retificar certas funcionalidades. Assim, basta consultar a tabela do histórico de versões e observar o que foi de novo acrescentado, dessa consulta pode-se, então, escolher a versão adequada ao que se procura, ver como exemplo a Tabela 10 de um histórico de versões de uma especificação.

Tabela 10 - Histórico de Versões de uma Especificação Funcional (exemplo)

Versão	Data	Pessoas	Ações
1.0	20-10-2017	Cláudia Filipa Ferreira Trigo	Responsável por escrever a especificação.
1.1	21-10-2017	Cláudia Filipa Ferreira Trigo	Descrição de alguns ecrãs das funcionalidades login e consulta de dados.
1.2	22-10-2017	Cláudia Filipa Ferreira Trigo	Reformulação do capítulo 1, os aspetos da funcionalidade login foram alterados para uma funcionalidade mais fácil de se perceber.

Depois de escrita a especificação, esta é enviada para o cliente e aguarda-se pela validação por parte do mesmo e, após isso, passa-se à resolução dessas funcionalidades.

Dentro da equipa de desenvolvimento, para além da existência da especificação, existe um manual técnico que tem informação sobre a empresa, a regra de negócio e como se instala o projeto pela primeira vez na máquina pessoal. Este manual é útil para a inserção de novos membros na equipa de desenvolvimento. Estes, assim que entram na equipa, a primeira coisa a fazer é lerem o respetivo manual para se inteirarem de como a aplicação funciona.

Existe também documentação feita pelo *tester,* após a sua entrada na equipa, neste caso a mestranda. Esta descreveu, passo a passo, como são feitos os testes de cada módulo existentes na aplicação. Este tipo de documentação também é muto útil e pertinente para a equipa de desenvolvimento, pois são registos que podem ser consultados futuramente na resolução de situações idênticas. Por exemplo, para a resolução futura de um *ticket* de um determinado módulo, poderá ser muito importante a consulta dos registos efetuados pela *tester*, aquando da concretização de uma determinada atividade de um módulo realizado num momento anterior, pois poderá ser suficiente reler a documentação especifica do módulo em si para se ficar a perceber rapidamente como fazer para reproduzir esse modulo

na aplicação.

3.4 Ambientes

Nesta secção irá ser apresentado os ambientes existentes na empresa onde a mestranda se encontra a realizar o estudo.

3.4.1 Ambiente DEV

A cada resolução de cada *ticket* é feito um *commit* e fica na própria máquina de desenvolvimento até que se faça *sync,* ou seja, ambiente DEV, nunca entrará no servidor onde está o projeto. Um *commit* só é feito quando o desenvolvedor tem a certeza que a resolução do *ticket* se encontra resolvida, ver a Figura 31 que é um exemplo de um *commit* feito no *Visual Studio*

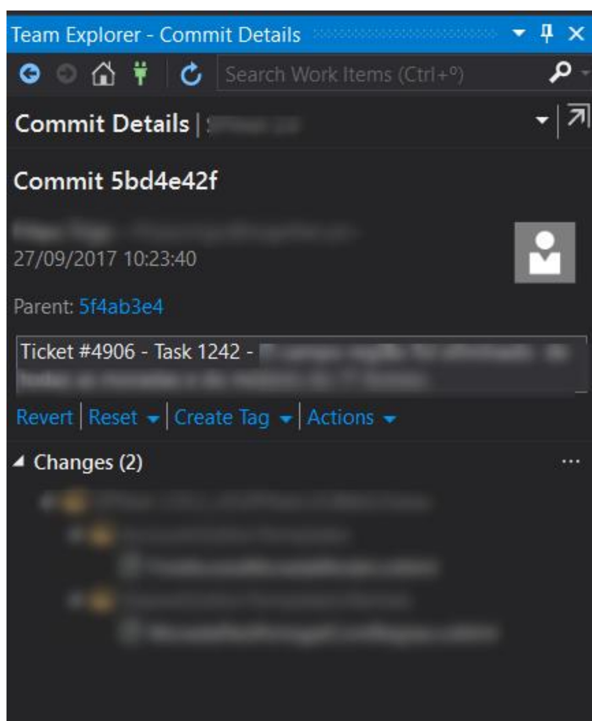


Figura 31 - Exemplo de commit

A cada resolução de um *ticket*, a pessoa que o solucionou tem que avisar o *tester* da equipa que já pode ir testar.

Este é o ambiente onde todos os elementos da equipa trabalham, inclusive o *tester*. Uma nota importante sobre este ambiente é que, para o *tester* fazer os testes aos *tickets* que forem resolvidos, terá que ter o seu ambiente e para isso existem alguns passos que se explicam mais abaixo.

1. Fazer *sync* no *Visual Studio*

Abrir o projeto no *Visual Studio* ir a *Team Explorer* e clicar em *sync*, ver Figura 32.

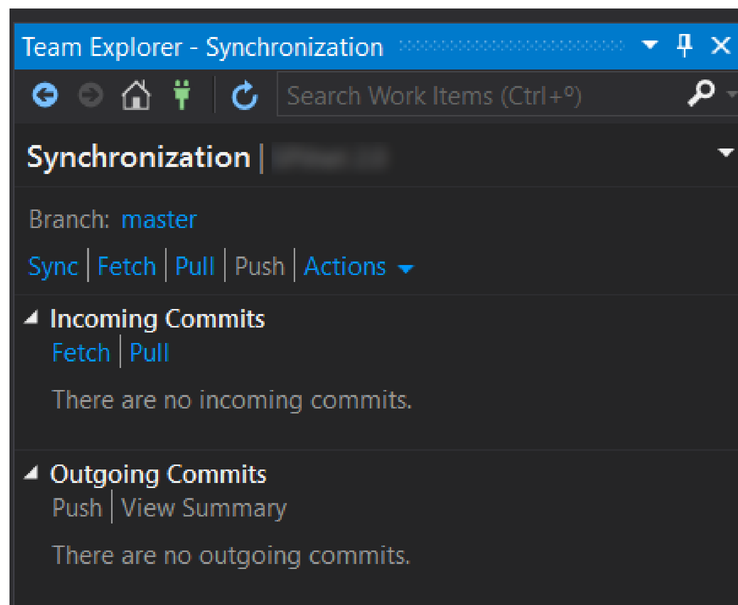


Figura 32 - Fazer Sync

2. Correr os Re-Runnables

Por vezes é preciso correr os *Re-Runnables,* que se encontram na pasta onde esta o projeto instalado com o nome *Re-Runnables Scripts,* ver Figura 33.

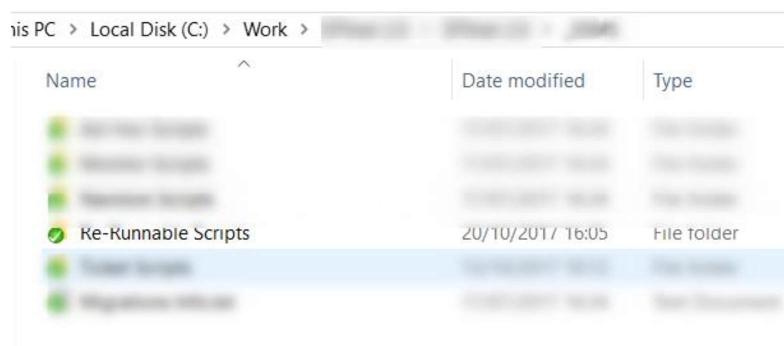


Figura 33 - Pasta dos Re-Runnables

Para executar alguns testes na aplicação e manter o projeto sempre atualizado, se a resolução dos *tickets* envolveu criação de scripts SQL, como inserção de dados em tabelas, criação de procedimentos, funções, *triggers* e vistas, para além de fazer *sync* terá de correr os *re-runnables* e bastará pressionar o ficheiro que diz respeito ao projeto ao ambiente DEV, ver Figura 34.

Figura 34 - Correr os re-runnables

3. Fazer *update-database*

Certas resoluções implicam a criação de novas tabelas ou adicionar novas colunas na base de dados, portanto o que tem que se fazer é ir ao *Package Manager* no *Visual Studio* escolher o projeto de acordo com a base de dados e escrever *update-database*, ver Figura 35.

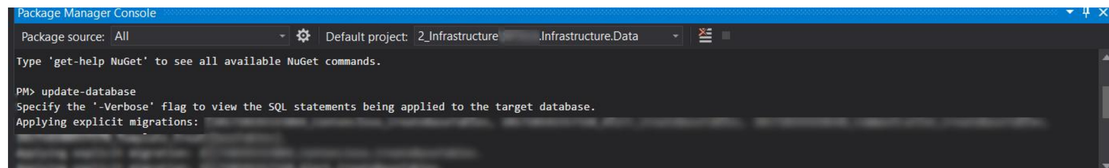


Figura 35 - Fazer update-database

Após o projeto estar atualizado, podemos por a correr a aplicação e o ambiente DEV têm o seguinte aspeto, ver Figura 36.

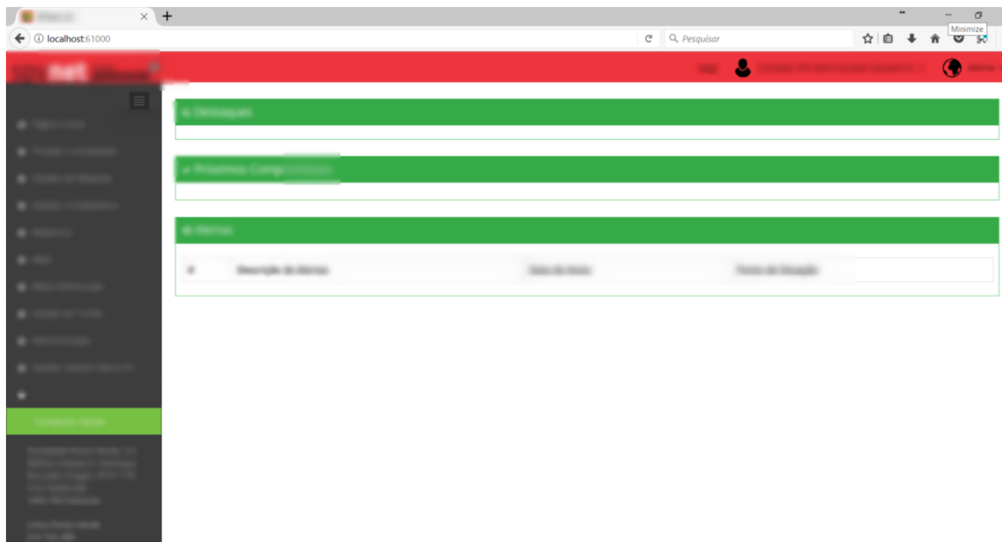


Figura 36 - Ambiente DEV

3.4.2 Ambiente PRE-QUA

Na empresa onde a mestranda se encontra a realizar o estudo, existe um ambiente dedicado somente ao de teste, ou seja, é uma réplica do ambiente de qualidade onde o cliente não tem acesso e serve unicamente para o *tester* efetuar os seus testes e tentar reproduzir os erros que são reportados nos tickets sem que haja danos na informação dos dados nos outros ambientes. Também serve para testar, verificar e atribuir os *tickets* que se encontravam a aguardar *deploy*, ver Figura 37.

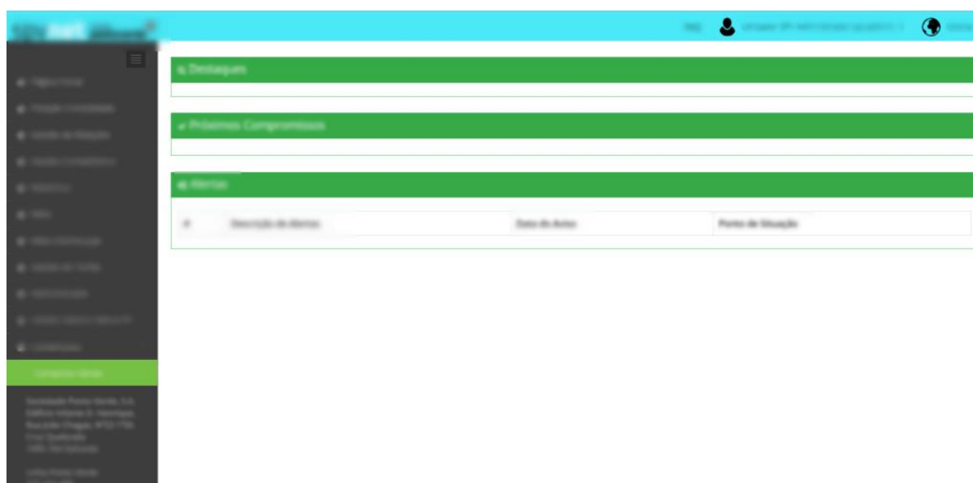


Figura 37 - Ambiente PRE-QUA

3.4.3 Ambiente QUA

Este ambiente é mais controlado e tipicamente encontra-se num servidor externo. Podem existir acessos por parte de *stakeholders* e é muito utilizado pelo cliente e pelo *stakeholders* para reportar erros da aplicação em tickets e é através dele que se fazem os testes e as validações dos *tickets,* ver Figura 38.

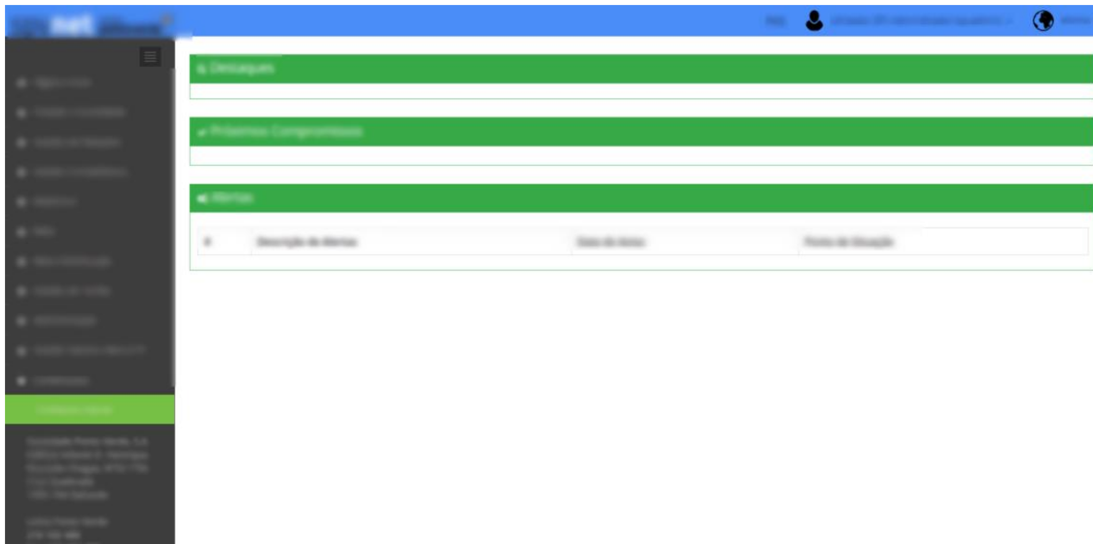


Figura 38 - Ambiente de QUA

Normalmente os elementos da equipa usam também este ambiente para verificar correções, apontando a máquina deles para a base de dados de QUA, assim podem ter acesso a determinados dados da base de dados que não existem na base de dados local. Normalmente a base de dados de QUA é muito parecida ou muito idêntica a base de dados de PRD:

3.4.4 Ambiente PRD

O ambiente de PRD é preciso ter especial atenção aos testes que são elaborados nele, pois este ambiente é o que cliente vê, não convêm ter erros visíveis. Raramente são feitos testes em PRD, após passagem de QUA a PRD os únicos que se fazem são testes de pesquisas e pouco mais, normalmente só para ver se a aplicação funciona bem, não fazendo com que a informação de cada cliente seja alterada, ver Figura 39.

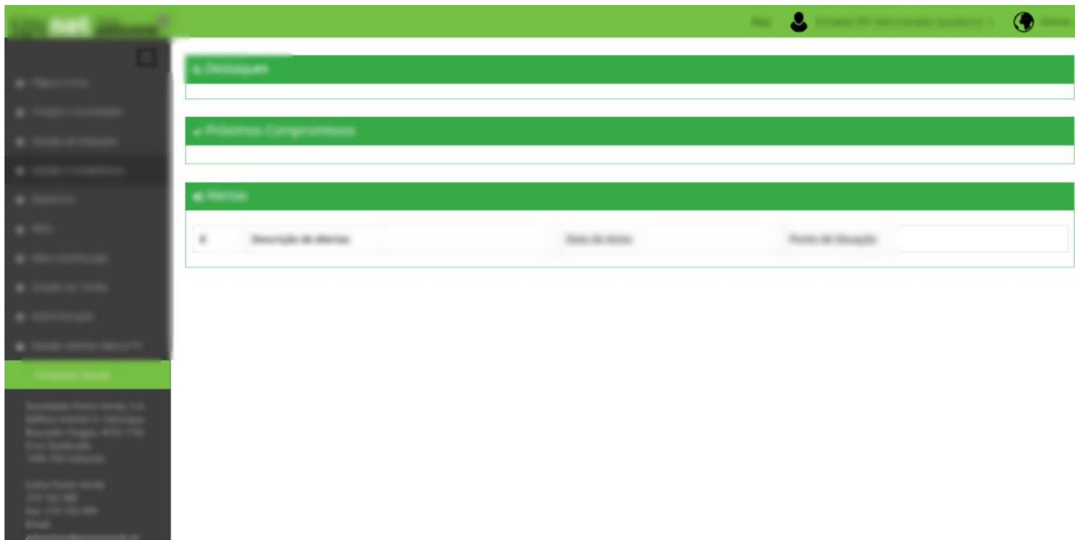


Figura 39- Ambiente PRD

Normalmente, se nas nossas máquinas temos a versão mais recente e foi reportado um ticket que o erro era em PRD, os *stakeholders* têm o direito de chegar junto da equipa de desenvolvimento e pedir uma certa funcionalidade, pois ainda não houve nova passagem a PRD. Esta situação é fácil de resolver, basta apontar a nossa máquina para a base de dados de PRD e os *stakeholders* ficam e observam com os seus olhos o que pretendem.

3.5 Elaboração dos testes

Para efetuar os testes aos *tickets*, o *tester* tem que ler o que esta descrito e o que é pedido e efetuar o suposto teste. Também é da tarefa do *tester* tentar reproduzir o erro que esta descrito no ticket e informar os desenvolvedores os passos de reprodução do erro, ver Figura 40 como exemplo de um ticket que foi reportado para a equipa resolver.

Figura 40 - Exemplo de um ticket

3.5.1 Reprodução do erro reportado

Na empresa muitos *tickets* são criados, semanas após semanas, com erros que vão aparecendo na aplicação e que precisam de ser resolvidos. O papel do *tester* neste caso é o seguinte:

- Ler a descrição do ticket;
- Analisar bem o erro reportado no ticket:
- Tentar reproduzir o erro uma, duas, três, as vezes que forem precisas até descobrir onde está;
- Erro descoberto e sabe reproduzir, o próximo passo é informar a pessoa da equipa responsável pela resolução desse *ticket*;
- Demonstrar onde está o erro com aplicação a correr localmente;

3.5.2 Testes aos *tickets* resolvidos

Existem sempre *tickets* para serem testados e validados, a mestranda sabe sempre quais os *tickets* que estão a aguardar teste, pois existe uma folha de Excel, já mencionada no capítulo da documentação, onde consta um filtro dedicado só ao *tester,* ver Tabela 11.

Tabela 11 - Filtro do tester

1199		4886	Ticket #4886 - Task 1199 -		Critica	Testado	QUA	2017-08-25
1200		4841	Ticket #4841 - Task 1200 -		Critica	Testado	QUA	2017-09-01
1203		4875	Ticket #4875 - Task 1203 -		Critica	Testado	QUA	2017-09-01
1207		4886	Ticket #4886 - Task 1207 -		Critica	Testado	QUA	2017-09-01
1210		4890	Ticket #4890 - Task 1210 -		Critica	Testado	QUA	2017-09-08
1214		4894	Ticket #4894 - Task 1214 -		Alta	Testado	QUA	2017-09-08
1217		4895	Ticket #4895 - Task 1217 -		URGENTE	Testado	QUA	2017-09-08
1220		4896	Ticket #4896 - Task 1220 -		Alta	Testado	QUA	2017-09-15
1221		4897	Ticket #4897 - Task 1221 -		Alta	Testado	QUA	2017-09-15
1222		4850	Ticket #4850 - Task 1222 -		Critica	Testado	QUA	2017-09-29
1223		4898	Ticket #4898 - Task 1223 -		Critica	Testado	QUA	2017-09-15
1224		4788	Ticket #4788 - Task 1224 -		Alta	Testado	QUA	2017-09-15
1231		4901	Ticket #4901 - Task 1231 -		Critica	Testado	QUA	2017-09-29
1232		4777	Ticket #4777 - Task 1232 -		Critica	Testado	QUA	2017-10-06
1234		4886	Ticket #4886 - Task 1234 -		Média	Testado	QUA	2017-09-22
1235		4625	Ticket #4625 - Task 1235 -		Critica	Testado	QUA	2017-09-22
1236		4618	Ticket #4618 - Task 1236 -		Critica	Testado	QUA	2017-10-13
1237		4866	Ticket #4866 - Task 1237 -		Critica	Testado	QUA	2017-10-13
1239		4903	Ticket #4903 - Task 1239 -		Alta	Testado	QUA	2017-10-06
1242		4906	Ticket #4906 - Task 1242 -		Alta	Testado	QUA	2017-09-29
1243		4907	Ticket #4907 - Task 1243 -		Alta	Testado	QUA	2017-09-29
1244		4908	Ticket #4908 - Task 1244 -		Critica	Testado	QUA	2017-09-29
1245		4909	Ticket #4909 - Task 1245 -		Critica	Testado	QUA	2017-10-20
1246		4910	Ticket #4910 - Task 1246 -		Alta	Testado	QUA	2017-09-29
1248		4569	Ticket #4569 - Task 1248 -		Critica	Testado	QUA	2017-10-06
1254		4898	Ticket #4898 - Task 1254 -		Critica	Testado	QUA	2017-10-06
1264		4926	Ticket #4926 - Task 1264 -		Alta	Testado	QUA	2017-10-13
1266		4928	Ticket #4928 - Task 1266 -		Critica	Testado	QUA	2017-10-20
1270		4788	Ticket #4788 - Task 1270 -		Média	Testado	QUA	2017-10-13
1271		4931	Ticket #4931 - Task 1271 -		Alta	Testado	QUA	2017-10-20
1275		4907	Ticket #4907 - Task 1275 -		Alta	Por Testar	DEV	2017-10-20
1276		4569	Ticket #4569 - Task 1276 -		Alta	Por Testar	DEV	2017-10-20
1277		4935	Ticket #4935 - Task 1277 -		Critica	Por Testar	DEV	2017-10-20
1280		4938	Ticket #4938 - Task 1280 -		URGENTE	Por Testar	DEV	2017-10-20

Pela análise do documento, pode-se concluir que existem, no final, alguns com estado o “testado” e “por testar”, logo cabe ao *tester* realizar o seu teste.

O *tester* lê o *ticket* e vai seguindo os passos consoante o que é descrito no *ticket* em causa. Para não existir nenhuma confusão quanto aos testes e aos passos do teste que já foram feitos, o *tester* tomou a iniciativa de elaborar uma folha de teste onde consta o registo de todos os passos que já foram realizados para um determinado teste, ver a Tabela 12 como exemplo.

Tabela 12 - Procedimento de teste

Número dos Passos	Passos de Teste	Descrição	Resultado	Comentário
1				OK
2				OK
3				OK
4				OK
5				OK
6				OK
7				OK
8				OK
9				OK

Pela análise podemos ver que existem cinco colunas no procedimento e teste que são:

- Número dos passos, ajuda a lembrar em qual passo se está em determinado *ticket*:
- Passos do teste, dizem respeito aos formulários da aplicação que ajudam para que não existam confusões
- Descrição, tem a ver com o que foi feito nesse teste;

- Resultado, é o que acontece com o que foi dito na descrição;

- O comentário tem uma resposta óbvia “OK” e encontra-se a verde, quer isto dizer que todos os testes que foram realizados passaram. Caso contrário, o comentário diria “NOT OK “e estaria a vermelho. Isto significa que é necessário verificar junto da pessoa que resolveu, onde se encontra e a causa desse problema;

Após o preenchimento dessa folha, terá de se avisar o responsável do desenvolvimento que se encontra testado e que passou ou não. Se passou está pronto para ir para QUA, se não passou cabe ao *tester* demonstrar o que fez para depois se perceber onde está a falha, sendo por vezes necessário criar um novo *ticket* com o erro que foi descoberto ao testar outro *ticket*, ver a Figura 41 de como se cria um novo *ticket*.

Figura 41 Criação de um novo ticket

Normalmente, é feita a passagem a QUA dos *tickets* que se encontram em DEV é regular, quem faz esta passagem costuma ser um elemento da equipa que não o *tester*. Quando já estiver pronto, faz-se chegar ao *tester* um ficheiro com o que foi para QUA, ver Figura 42.

```
6  # Change Log #
7
8  | Commit  | Descrição
   |
9  | ------- | ---------------------------------------------------------------------------------
   ----------------------- |
0  | bc4e917 | Ticket #4681 - (WIP) Esconde os debuggers
1  | 457ac34 | Ticket #4618 - Task 1236 - Fix, após e-mail de confirmação no fluxo PA ser
   preenchido e guardado, não é possível navegar novamente para esse mesmo ecrã.
2  | 13389dd | Ticket #4928 - Task 1266 - Fix Validação do E-mail da factura electrónica.
3  | 334ec31 | Ticket #4904 - Task 1240 - Inclui resolução de "Merge conflit"
4  | 886abb5 | Ticket #4930 - Task 1269 - Fix Commit anterior. Força a criação duma morada
   sede caso esta esteja a null.
5  | 646b63a | Ticket #4875 - Task 1238 - Após reunião com a equipa decidiu-se fazer rollback
   ao código já commited e junto do RMP validar e tentar especificar as regras de negócio para
   cancelar e reativar declarações. (cont.)
6  | 23a758b | Ticket #4926 - Task 1264 - Criação de diagramas
7  | 6f04bca | Ticket #4875 - Task 1238 - Após reunião com a equipa decidiu-se fazer rollback
   ao código já commited e junto do RMP validar e tentar especificar as regras de negócio para
   cancelar e reativar declarações.
8  | c94721d | Ticket #4930 - Task 1269 - Fix PA Ecrã de Morada de Sede agora é sempre
   apresentado e preenchido. Ticket #4788 - Task 1270 - Fix Opção da dropdown é seleccionado
   apenas manualmente.
9  | da733e0 | Ticket #4914 - Task 1251 - Criação dos Controllers + Views + Models para o
   módulo Contencioso
0  | d47362b | Ticket #4618 - Task 1236 - Fix - Fluxo PA recomeça no step "Moradas" apesar de
   o utilizador ter navegado para o 1º step (confirmação de E-mail) anteriormente.
1  | 71e7f82 | Ticket #4916 - Task 1255 - Erro ao Arquivar Contrato - Âmbito Parcial
2  | 8832d78 | Ticket #4866 - Task 1237 - Âmbito Parcial - Regras da Data do menu Arquivo
3  | 764e53e | Ticket #4898 - Task 1254 - Âmbito Parcial - Entrega de Declaração - Alertas
   Errados
```

Figura 42 -Ficheiro com o conteúdo de QUA

Após esta informação chegar ao *tester,* o próximo passo será a validação e atribuição dos *tickets* aos *stakeholders,* mas antes de se começarem a efetuar testes em PRE QUA, o *tester* tem de ir comparar a informação do ficheiro com os *tickets* para se verificar se é igual àquela que existe no VST, separador que contém essa informação e que se chama de aguardar *deploy*, ver Figura 43.

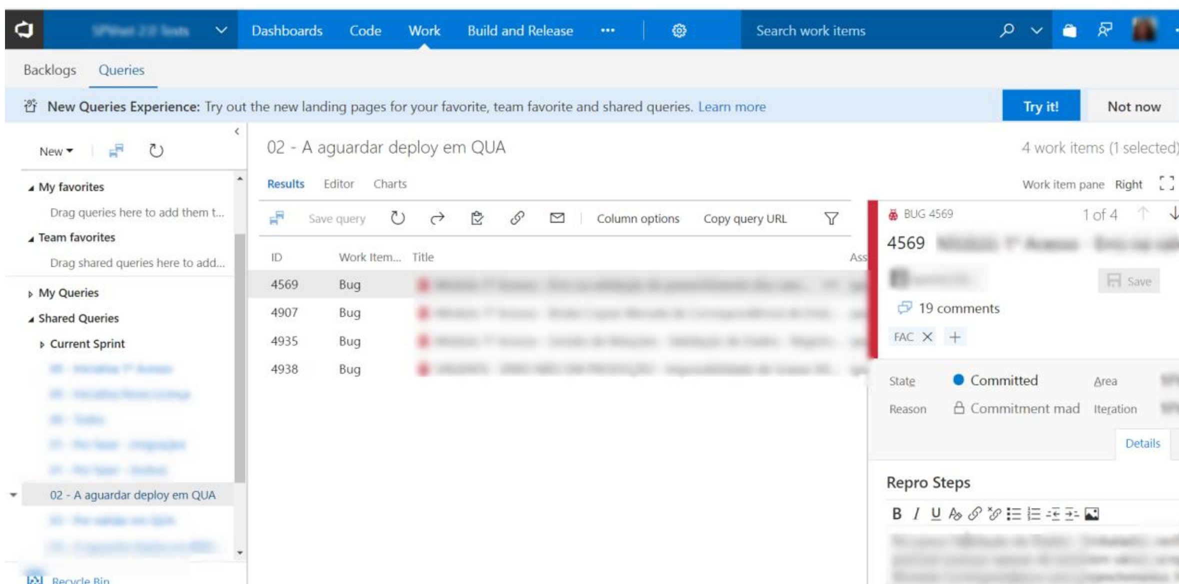


Figura 43 - Aguardar deploy em QUA

Seguidamente, após esta verificação, o próximo passo é o teste. O teste é feito em PRE QUA, se a verificação correu bem, passa-se para atribuição, ou seja, atribui-se a pessoa que reportou o ticket e escreve-se a seguinte mensagem "Foi feito *deploy* em QUA", ver a Figura 42 como exemplo.

4908 Módulo 1º Acesso - Botões de fim de página - Presença e Cor
Unassigned 3 comments Add tag Save Follow
State Done Area Tests Updated 1
Reason Work finished Iteration Tests\DONE AUG-DEC 2016
Details (1)
Add a comment. Use # to link a work item or @ to mention a person
Stakeholder 1 commented a week ago
Validado em QUA.
Filipa Trigo commented a week ago
Foi feito deploy em QUA.
Filipa Trigo commented 4 weeks ago
O botão sair já está presente em todas as páginas da aplicação de atualização de dados.
O botão sair e voltar já se encontram com a cor branca e os restantes botões a verde.
Aguarda deploy

Figura 44 - Mensagem de atribuição e validação por parte dos stakeholders

Neste ticket podemos ver que a informação dada pelo *stakeholder* 1 é que o *ticket* já se encontra validade em QUA e com o estado DONE, ou seja, este *ticket* já se encontra fechado.

3.6 Análise do impacto dos testes

A cada *deploy* existem *tickets* que são aprovados e outros que não são aprovados, pois a sua resolução não se encontra como pedido na descrição do *ticket.* Esta informação também pode ser vista no VST, bastando apenas clicar no menu do lado esquerdo, ver Figura 45.

Figura 45 – Voltaram para trás

Com a resolução de *tickets* e com a passagem a QUA ou a PRD é normal que algum *ticket* não se encontre como consta no requisito da especificação ou como foi reportado no *ticket*, então os *stakeholders* vêm-se obrigados a mandar esses *tickets* de volta para os programadores.

De seguida será demonstrado um exemplo do um *ticket* que voltou para trás com a funcionalidade mal desenvolvida, ver Figura 46

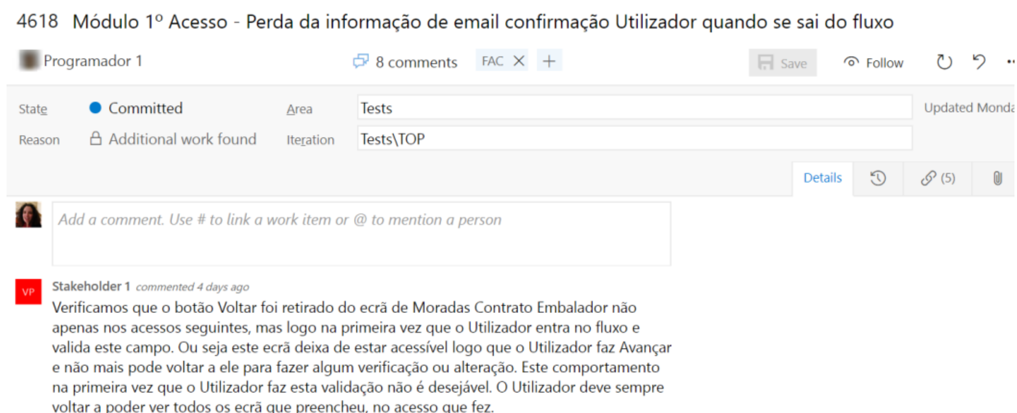


Figura 46 - Ticket que voltou para trás (exemplo)

Na fase anterior á entrada na empresa da mestranda, com funções de *tester*, as práticas de trabalho eram propicias a surgir lacunas no código desenvolvido, muitas funcionalidades estavam inacabadas ou passavam para PRD sem garantia de que o *software* foi bem testado ou mesmo sem funcionar.

Ainda, os *deploys* em QUA ou em PRD eram pouco frequentes. Quando existia a passagem a QUA, agrupava muitos desenvolvimentos e por vezes nem eram devidamente testados por terem tantos desenvolvimentos agrupados.

No gráfico da Figura 47, encontram-se representados os *deploys* com os respetivos *tickets* não aprovados, no período anterior á entrada na *tester* na equipa. Pela análise ficamos a saber que o número de *tickets* que voltou para trás se encontra entre os 50 e os 70%, esta situação é preocupante, em média 12 *tickets* voltaram para trás e apenas 9 *deploys* foram feitos.

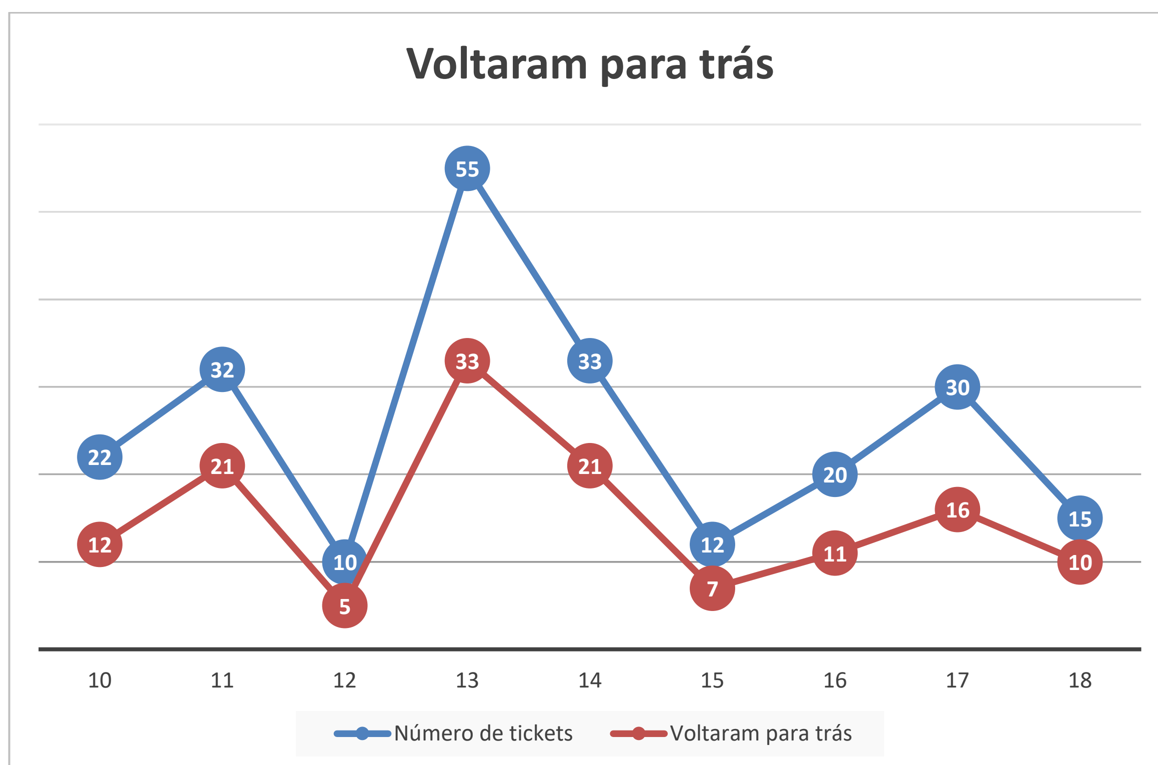


Figura 47 - Tickets que voltaram para trás

Perante este cenário, a equipa de desenvolvimento reuniu e chegou á conclusão que devia existir um elemento dedicado aos testes de cada *ticket*, ou seja, um *tester* que testa o que é resolvido em ambiente DEV e que os *deploys* deviam ser mais regulares e não agruparem muitas resoluções de *tickets* num só *deploy*.

A situação da resolução de *tickets* feita pela equipa está a correr bem e prevê-se que ainda vai correr melhor daqui para a frente.

Nota-se uma grande melhoria quando comparado com cenário anterior. Isto é francamente positivo para a empresa, é sinal que a equipa de desenvolvimento se encontra a fazer bem a resolução dos *tickets* e que estão a ser definitivamente testados como deve ser, antes de ir para QUA ou PRD.

Na empresa onde a mestranda se encontra a realizar o estudo, para além das prioridades dos tickets que são para cumprir, caso haja algum que voltou atrás, a prioridade passou a ser esse *ticket.*

Com análise do gráfico da Figura 48, podemos perceber que houve uma diminuição dos *tickets* a voltar para trás, ou seja, em média existem 2 *tickets* a voltar para trás com 9 *deploys* feitos. Isto é muito positivo para equipa de desenvolvimento, pois significa que o *tester* está a fazer bem o seu trabalho, tendo testado exaustivamente a resolução dos *tickets* e que as funcionalidades descritas nos *tickets,* na sua maioria, já não se encontram com erros ou quebradas a meio.

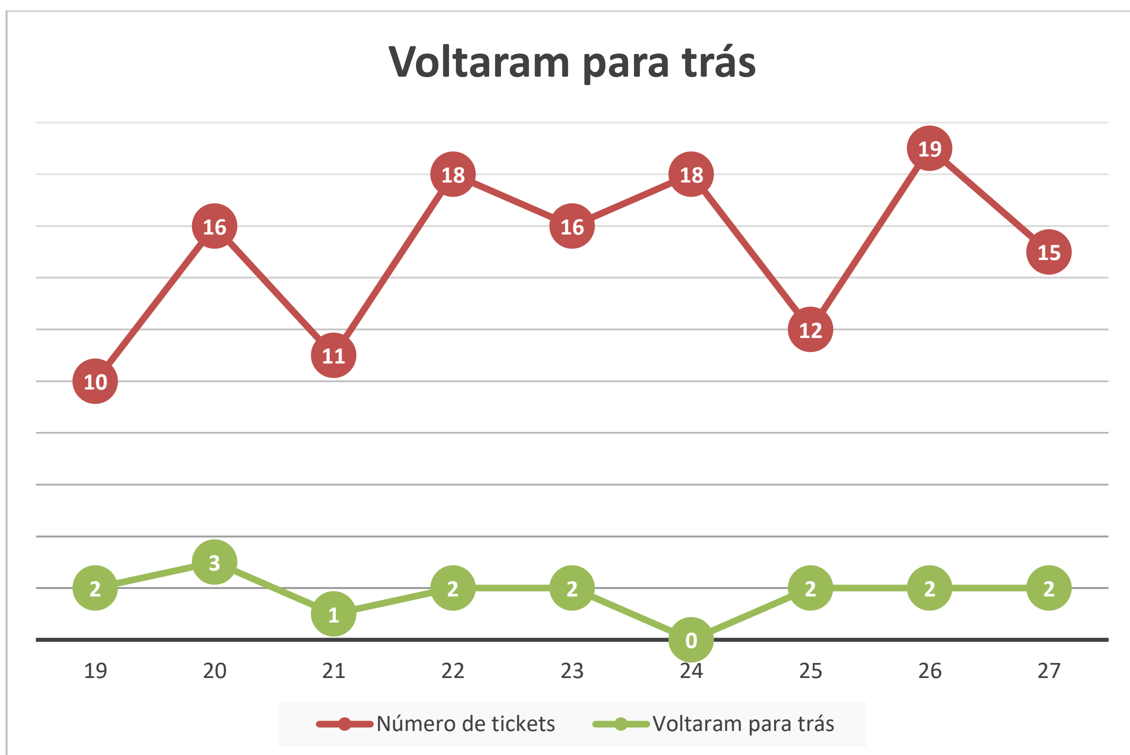


Figura 48 - Tickets reprovados

Em jeito de resumo dos dados já apresentados, o gráfico da Figura 49, permite uma panorâmica geral da evolução do comportamento da empresa em função da realização de testes de software. A linha vertical representa a divisão temporal compreendida nos períodos anteriores e posteriores á inclusão do elemento dedicado aos testes ao *software* desenvolvido pela empresa em estudo.

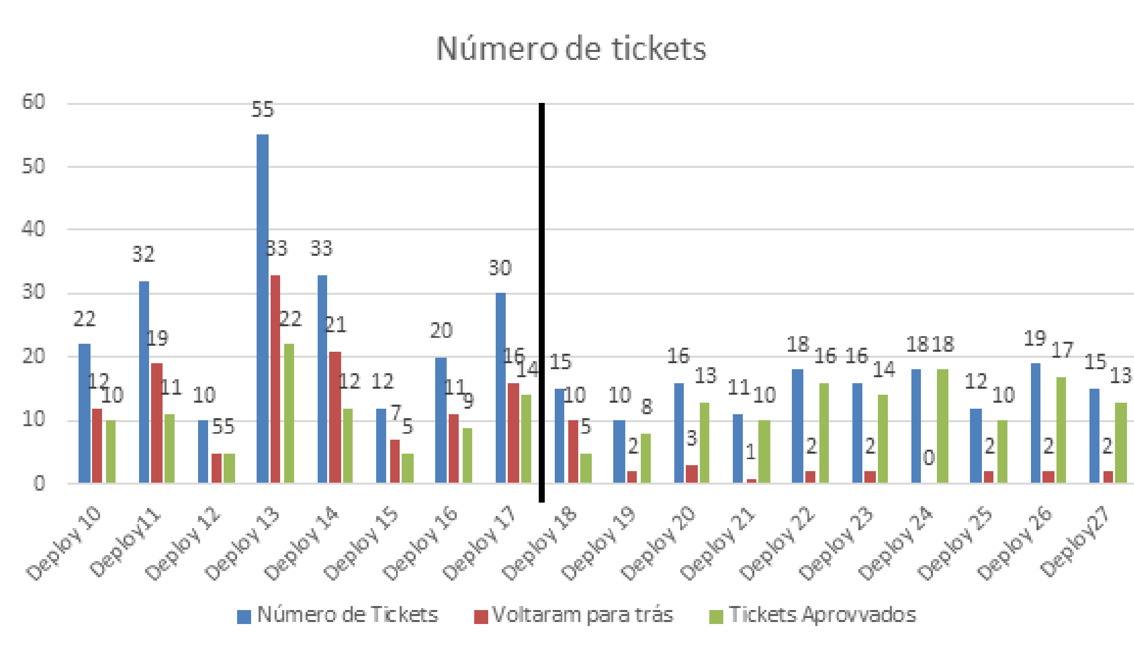


Figura 49 - Número de tickets

Pela análise conclui-se que a partir do *deploy* número 18 houve uma descida dos *tickets* a voltar para trás, assim sendo, comprova-se que o papel do *tester* é relevante, pois serve para garantir que as resoluções dos *tickets* que se encontram pedidas nas descrições dos *tickets* ou nas especificações existentes na documentação da equipa, deixem de existir e evitar que após a passagem a QUA ainda surjam erros.

Consegue-se ter um histórico de como era antes de existir um elemento dedicado aos testes. Conclui-se que havia um número reduzido de *tickets* que eram aprovados ou fechados, assim, pode afirmar-se que esta situação talvez tenha acontecido por falta de testes, uma vez que os programadores da equipa não tinham tempo para os realizar e acreditavam que o seu código estava 100% correto e a funcionar. Após a inserção de um *tester*, o número de *tickets* a serem aprovados subiu consideravelmente, o que comprova que o *tester* é um elemento imprescindível para que a equipa tenha alcançado o sucesso verificado devido ao número de *tickets* que deixou de voltar para trás e a serem fechados ou aprovados.

Quantos mais testes forem realizados, mais depressa se encontram erros nos *tickets* atempadamente, ou seja, antes de irem para QUA e os *stakeholders* evitam contratempos, pois deixam de precisar de ser eles a detetarem esses erros, facilitando, assim, o seu desempenho em beneficio da empresa e de todos os intervenientes.

4. Conclusão

Os testes de *software* têm-se revelado cada vez mais importantes para as empresas e o seu progresso tem seguido no sentido de cada vez mais ser uma tarefa independente e imparcial em relação à tarefa de programação de uma aplicação.

À medida que o conceito de testes de *software* se foi construindo, surgiram técnicas e métodos de forma a tornar esta tarefa mais eficiente, ajudando as equipas de desenvolvimento a terem o tão desejado sucesso com a aplicação que se encontram a construir.

Nesta dissertação foram aprofundados os conceitos de testes aplicacionais, testes funcionais, testes unitários, testes através da especificação, a inclusão de um elemento somente dedicado aos testes e, quão importante é a documentação dentro do seio da equipa, pois o ter tudo documentado pode ser uma ferramenta de salvação quando alguma coisa corre menos bem.

Ainda, foi realizado um estudo que visa evidenciar como é feita a gestão de *tickets* dentro da empresa onde a mestranda se encontra e como ocorre a atribuição, validação, aprovação e fecho dos *tickets*.

Com este estudo considera-se que fica reforçada a ideia de que um *tester* é uma peça fundamental no seio de uma equipa de desenvolvimento, pois este elemento contribuir para que mais depressa se detetem os erros ou anomalias, poupando recursos às empresas.

5. Referências

Acharya, S., & Pandya, V. (2012). Bridge between Black Box and White Box – Gray Box. *International Journal of Electronics and Computer Science Engineering*, pp. 175-185.

Andrade, N. I. (2016). *Modelagem de Requisitos de Software*. Obtido de slideplayer: http://slideplayer.com.br/slide/11346694/

Arora, & Sinha. (2 de Fevereiro de 2002). Web Application Testing: A Review on. *International Journal of Scientific & Engineering Research, Volume 3*, pp. 1-6.

Bahia, G. B. (2015). *MAKE: UM FRAMEWORK PARA GERAÇÃO DE DADOS PARA TESTES UNITÁRIOS EM JAVA*. Obtido de SlidePlayer: http://slideplayer.com.br/slide/3971008/

Barbosa, E. F. (2009). *Introdução ao Teste de Software.*

Bartié, A. (2002). *Garantia da Qualidade de Software* (5 ed.). Elsevier Editora. Obtido em 08 de 10 de 2017

Basic software testing v2.20. (05 de 02 de 2015). Obtido de SlideShare: https://www.slideshare.net/WarayuthWongpaiboonw/ais-basic-software-testing-v220

Bedani , J. (2017). *Engenharia de Software 2 - Técnicas para levantamento de Requisitos*. Obtido de devmedia: http://www.devmedia.com.br/engenharia-de-software-2-tecnicas-para-levantamento-de-requisitos/9151

Benadi, J. (2017). *Engenharia de Software 2 - Técnicas para levantamento de Requisitos*. Obtido de devmedia: http://www.devmedia.com.br/engenharia-de-software-2-tecnicas-para-levantamento-de-requisitos/9151

Carreira, L. (2014). *A Linguagem de Especificação Z.* Obtido de slideplayer: http://slideplayer.com.br/slide/384264/

Carvalho, M. F. (2010). *Automatização de Testes de Software - Dashboard QMSanalyser*. Coimbra.

Costa, C. (29 de Setembro de 2001). *Instrodução a Teste de Software.* Obtido de Slideshare: http://pt.slideshare.net/x25treinamento/introduo-ao-teste-de-software

Costa, S. L., & Gonçalves, V. P. (2012). *Especifcação Formal de Software.* São Paulo: Universidade de São Paulo.

Crespo, A. (2004). *Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo.* Obtido de researchgate.net: https://www.researchgate.net/profile/Adalberto_Crespo/publication/237497188_Uma_Metodologia_para_Teste_de_Software_no_Contexto_da_Melhoria_de_Processo/links/54e5d1040cf2cd2e028b338b.pdf

crvs-dgb. (s.d.). Obtido de Define System Requirements: http://www.crvs-dgb.org/en/activities/analysis-and-design/8-define-system-requirements/

Define Target System Architecture. (s.d.). Obtido de crvs-dgb: http://www.crvs-dgb.org/en/activities/analysis-and-design/8-define-system-requirements/

Eliza, F., & Lagares, V. (18 de 09 de 2017). *Processo de Teste de Software*. Obtido de devmedia: http://www.devmedia.com.br/processo-de-teste-de-software/23795

Ellauri, J. (s.d.). *Functional Testing*. Obtido em 15 de 10 de 2017, de abstracta: http://abstracta.com.uy/pt/servicos-consultoria-outsourcing/teste-funcionais

Engholm, H. J. (2013). *Engenharia de Software na Prática.* Novatec . Obtido em 08 de 10 de 2017

Espinha, R. G. (07 de 03 de 2016). *Qual a importância da documentação em projetos?* Obtido de artia: http://artia.com/blog/qual-a-importancia-da-documentacao-em-projetos/

estamosjuntos. (s.d.). Obtido de quem somos: http://estamosjuntos.pt/quem-somos/

Farias, G. (2017). *Udemy.* Obtido de Testes Automáticos + Curso Completo de Teste de Software.

Ferreira, R. (2010). *Desenvolvimento, Testes e Qualidade de Software.* Lisboa: Universidade Lusófona de Humanidades e Tecnologias.

Figueiredo, C., Neve, J., Magalhães, L., & Pinto, V. (2002). *Especificação Formal de Software.* Porto.

Gomes, F. (2014/2015). *A importância dos testes para a qualidade do software*. Obtido de DEVMEDIA: http://www.devmedia.com.br/a-importancia-dos-testes-para-a-qualidade-do-software/28439

IEEE. (16 de 12 de 1998). *829-1998 - IEEE Standard for Software Test Documentation*. Obtido de IEEE Xplore - Digital Library: http://ieeexplore.ieee.org/document/741968/

Janssen, D., & Janssen, C. (2017). *Mobile Application Testing*. Obtido de techopedia: https://www.techopedia.com/definition/30672/mobile-application-testing

Jesus, S. C. (2013). *Métodos de testes aplicacionais.* Universidade de Lisboa: Instituto Superior de Economia e Gestão. Obtido de https://www.repository.utl.pt/bitstream/10400.5/6398/1/DM-SICJ-2013.pdf

Junior, E. T. (2007). *Processos de Teste de Software.* Itatiba – São Paulo – Brasil.

Júnior, J. N. (24 de 01 de 2014). *Tecnicas de Especificacao de Testes*. Obtido de 4alltests: http://4alltests.webs.com/apps/blog/show/41009078-tecnicas-de-especificacao-de-testes

Khan, E., & Khan, F. (3 de 11 de 2012). A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. *International Journal of Advanced Computer Science and Applications*, pp. 1-4.

Koshy, V. (2005). *Action Research for Improving Practice - A practical guide.* Londres: Paul Chapman Publishing.

Leal, R. (03 de 2008). *4 Teste Funcional.* Obtido de Maxwell: https://www.maxwell.vrac.puc-rio.br/12322/12322_5.PDF

Lopes, J. M. (30 de Outubro de 2009). *Testes Funcionais.* Obtido de SlideShare:

https://pt.slideshare.net/julianamarialop/testes-funcionais

Macoratti, J. C. (2012). *Conceitos : Especificação de requisitos*. Obtido de macoratti.net: http://www.macoratti.net/07/12/net_fer.htm

Maldonado, J. C. (2004). *Instrodução ao teste de software* . Obtido de nabble.com: http://pbjug-grupo-de-usuarios-java-da-paraiba.1393240.n2.nabble.com/attachment/2545944/0/Introducao%20Ao%20Teste%20De%20Software%20-%20Maldonado,Jose.pdf

Mendes, A. (2016). *Plano de Teste - Um Mapa Essencial para Teste de Software*. Obtido de DEVMEDIA: http://www.devmedia.com.br/plano-de-teste-um-mapa-essencial-para-teste-de-software/13824

Meriat, V. (28 de Janeiro de 2013). *Testes e mais testes... O porquê dos testes de software* . Obtido de vitormeriat: http://www.vitormeriat.com.br/2013/01/28/testes-e-mais-testes-o-porqu-dos-testes-de-software/

Neto, A. C. (2014). Engenharia de Software Magazine. *Introdução a Teste de Software*, p. 54.

Patil, K. (05 de 04 de 2017). *WHAT IS BLACK BOX TESTING?* Obtido de bitwaretechnologies: https://bitwaretechnologies.com/difference-black-box-white-box-testing/

Paulo, J. (20 de 02 de 2015). *A importância dos testes unitários*. Obtido de atmdigital: https://www.atmdigital.com.br/home

Pereira, V. M. (2011). *Vantagens e/ou Desvantagens do.* Lisboa: Instituto Universitário de Lisboa.

Rongala, A. (24 de 04 de 2015). *What is White Box Software Testing: Advantages and Disadvantages*. Obtido de invensis: https://www.invensis.net/blog/it/white-box-software-testing-advantages-disadvantages/

Rouse, M. (2017). *Definition hotfix*. Obtido de techtarget: http://searchwindowsserver.techtarget.com/definition/hotfix

Santiago, I. (02 de 05 de 2010). *Automação de Teste Funcionais - Selenium.* Obtido de SildeShare: https://pt.slideshare.net/powerirs/automao-de-teste-funcionais-selenium-3936751

Santos, J. (11 de Junho de 2016). *Testes Unitários* . Obtido de slideshare: https://pt.slideshare.net/jadsonjs/testes-unitarios-62949111

Silva, P., Alves, T., & Andrade, E. (2015). REVISTA DE CIÊNCIAS EXATAS E TECNOLOGIA. *AUTOMAÇÃO DE TESTES FUNCIONAIS*, p. 116.

Silva, W. (Julho/Dezembro de 2012). Ferramentas free para teste de software. *Universitas Gestão e TI*, p. 59.

Simões, G. R. (2014). *Automação da análise de qualidade.* Coimbra.

Software Application Testing. (2013). Obtido de ideainfinityllc: http://www.ideainfinityllc.com/software-application-testing.html

Sousa, S. (2015). *Abordagem Estratégica ao Teste de Software*. Obtido de sildeplayer: http://slideplayer.com.br/slide/385708/

Tercete, A. (28 de 06 de 2013). *Por que você não escreve Testes Unitários?* Obtido de SlideShare: https://pt.slideshare.net/alextercete/testes-unitrios

Thiago, A. (06 de 04 de 2011). *Testes Unitários: por que escrever?* Obtido de codeatest: http://www.codeatest.com/testes-unitarios-porque-escrever/

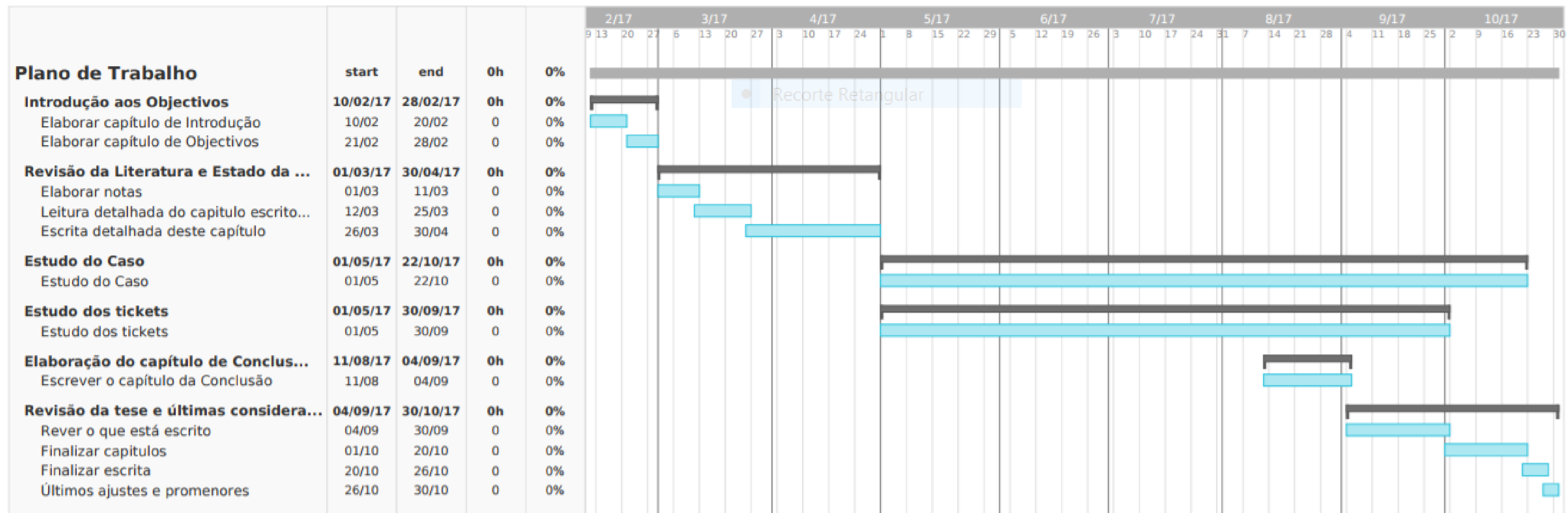
Tonsing, S. L. (2008). *Análise e Especificações de Requisitos.* Ciência Moderna. Obtido em 25 de 03 de 2017, de linhadecódigo: http://www.linhadecodigo.com.br/artigo/3224/analise-e-especificacoes-de-requisitos.aspx

Trust , T. (2015 de Abril de 2015). *targettrust*. Obtido de Os 13 principais tipos de Testes de Software!: https://targettrust.com.br/blog/os-13-principais-tipos-de-testes-de-software/

Ventura , P. (11 de 05 de 2016). *O que é Requisito Funcional*. Obtido de ateomomento: http://www.ateomomento.com.br/o-que-e-requisito-funcional/

Yin, R. (2013). *Case Study Research: Design and Methods* (5ª ed., Vol. Applied Social Research Methods). SAGE Publications.

ANEXO A: Trabalhos realizados



Plano de Trabalho	start	end	0h	0%
Introdução aos Objectivos	**10/02/17**	**28/02/17**	**0h**	**0%**
Elaborar capítulo de Introdução	10/02	20/02	0	0%
Elaborar capítulo de Objectivos	21/02	28/02	0	0%
Revisão da Literatura e Estado da ...	**01/03/17**	**30/04/17**	**0h**	**0%**
Elaborar notas	01/03	11/03	0	0%
Leitura detalhada do capitulo escrito...	12/03	25/03	0	0%
Escrita detalhada deste capítulo	26/03	30/04	0	0%
Estudo do Caso	**01/05/17**	**22/10/17**	**0h**	**0%**
Estudo do Caso	01/05	22/10	0	0%
Estudo dos tickets	**01/05/17**	**30/09/17**	**0h**	**0%**
Estudo dos tickets	01/05	30/09	0	0%
Elaboração do capítulo de Conclus...	**11/08/17**	**04/09/17**	**0h**	**0%**
Escrever o capítulo da Conclusão	11/08	04/09	0	0%
Revisão da tese e últimas considera...	**04/09/17**	**30/10/17**	**0h**	**0%**
Rever o que está escrito	04/09	30/09	0	0%
Finalizar capitulos	01/10	20/10	0	0%
Finalizar escrita	20/10	26/10	0	0%
Últimos ajustes e promenores	26/10	30/10	0	0%

ANEXO B: *Tickets*

B.1 Numero de *tickets* num *deploy*

Pela análise do gráfico da Figura 50, podemos concluir que o número de *tickets* por *deploy* já não é tão elevado como o verificado no subcapítulo anterior. Pela análise pode-se verificar que *deploy* número 13 obteve um conjunto de tickets 55 o que não é bom uma vez que assim fica uma tarefa difícil de testar este elevado de número. A partir do *deploy* número 19 houve uma melhoria pois os *deploys* são muito mais regulares e assim é muito mais fácil detetar erros ou funcionalidades que se encontrem quebrados. Porque, a cada *deploy* pode-se fazer testes exaustivos tanto no ambiente de desenvolvimento como após a passagem QUA os testes feitos pelo *tester* no ambiente de teste, PRE-QUA.

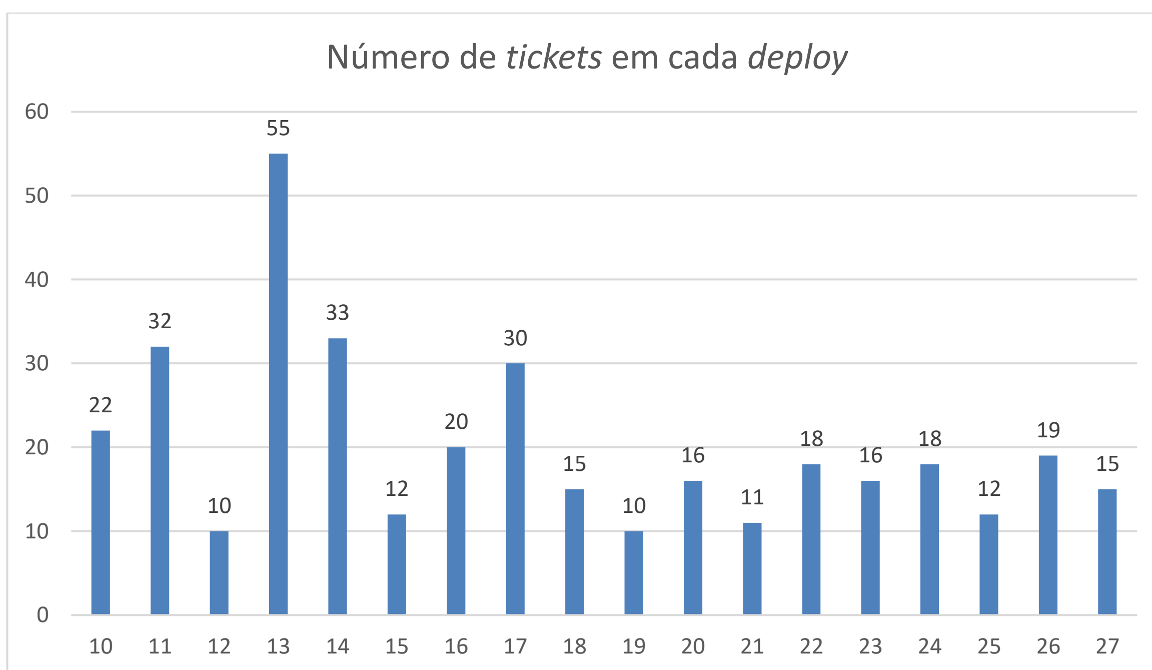


Figura 50 - Número de tickets em cada deploy

B.2 Número de tickets aprovados sem *tester*

Com a análise do gráfico da Figura 51 ficamos a perceber qual o número de *tickets* que foram aprovados.

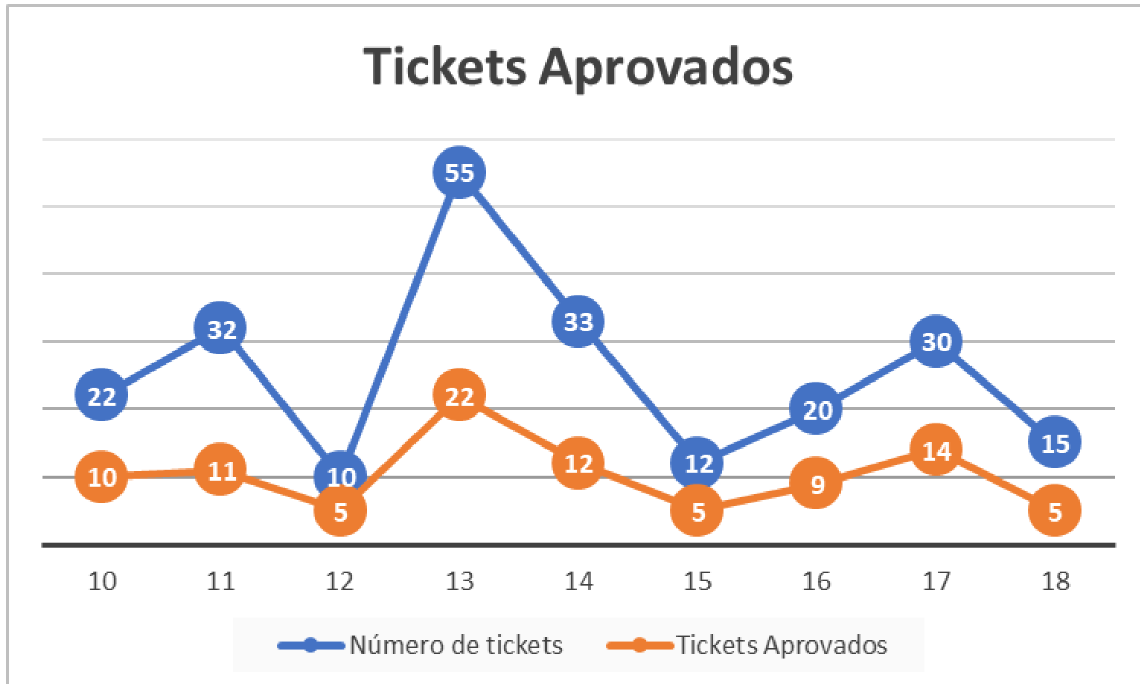


Figura 51 - Número de tickets aprovados antes

B.3 Número de *tickets* aprovados com *tester*

Com a analise do gráfico da Figura 52, podemos perceber a quantidade *tickets* que se encontram aprovados pelos *stakeholders*. Nota-se uma grande melhoria quando comparado com cenário anterior. Isto é francamente positivo para a empresa, é sinal que a equipa de desenvolvimento se encontra a fazer bem a resolução dos *tickets* e que estão a ser definitivamente testados como deve ser, antes de ir para QUA ou PRD.

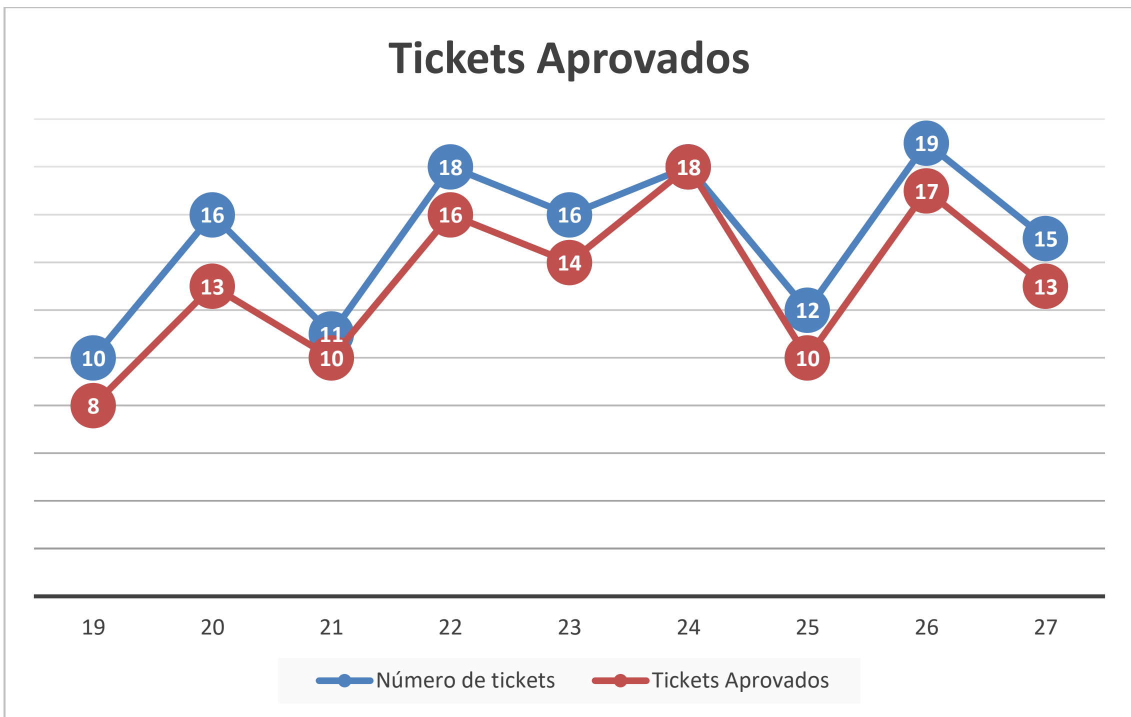


Figura 52 - Tickets Aprovados