Cátia Sousa Lopes

Automação de testes e o impacto no desenvolvimento de software

Dezembro 2020

Instituto Politécnico de Viseu
Escola Superior de Tecnologia e Gestão de Viseu
IPV - ESTGV | 2020
Automação de testes e o impacto no desenvolvimento de software
Cátia Lopes

limite texto

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

Cátia Sousa Lopes

Automação de testes e o impacto no desenvolvimento de software

Tese de Mestrado

Sistemas e Tecnologias de Informação para as Organizações

Professor Doutor Filipe Sá

Dezembro de 2020

RESUMO

Nos dias de hoje, são imensas as tarefas nas quais o *software* (SW) desempenha um papel crítico, o que, num futuro próximo, continuará a ser verificado, dada a exponencial utilização das Tecnologias de informação (TI).

A capacidade de fidelização do utilizador define-se pela excelência do SW. Quanto menos falhas este tiver, maior será a probabilidade de conseguir fidelizar o utilizador.

Para que essa situação seja possível, os testes são cruciais na etapa do desenvolvimento, uma vez que estes têm como principal propósito a identificação de erros.

Atualmente, empresas ou pessoas individuais que desenvolvem SW já se deparam com um problema: o aumento da complexidade decorrente da adição de funcionalidades de forma exponencial. Tal, conduz a uma impossibilidade de testar todo o SW, de cada vez que é objeto de alguma alteração. Estas alterações podem provocar falhas em funcionalidades que já existiam há algum tempo e estavam estabelecidas como funcionalidades estáveis. Por esse motivo, eram apenas testadas as novas funcionalidades e alterações. Para tentar resolver este problema, uma boa abordagem seria apostar na implementação de testes automatizados, sem esquecer também a importância e necessidades dos testes manuais.

Os testes automatizados, podem ser implementados com o objetivo de validar a aplicação, como um todo. Assim sendo, terão de ser definidos os testes *core*, de modo a detetar falhas quando existe alguma alteração ao SW. Desta forma, garante-se que uma funcionalidade, que não foi alterada continua a ter o comportamento desejável.

Nos primórdios do desenvolvimento da automação de testes colocavam-se duas questões: “Como vamos executar estes testes?”; “É necessário o fator humano para realizar esta operação?”. Ora, tais questões têm já solução. Atualmente, não é necessário que um humano execute os testes automatizados que foram desenvolvidos, sendo que, a palavra “automatizados”, por si só, significa que, tais testes são executados sem intervenção humana.

Este trabalho está inserido no âmbito de projeto final referente ao mestrado em Sistemas e Tecnologias de Informação para as Organizações, e tem como objetivo realizar uma análise sobre os testes automatizados e manuais, e os benefícios da automação no desenvolvimento de SW, permitindo selecionar quais as ferramentas que melhor se adequam num caso real. Foram validadas várias ferramentas, de modo a investigar se são efetivas para

encontrar erros. O projeto foi desenvolvido na empresa SIBS, enquadrado na equipa do IPS - *Instant Payments Solution* (serviço de transferências instantâneas).

Em termos práticos, este projeto está focado na implementação de uma bateria de testes automatizados de modo a encontrar falhas no SW. Assim é possível validar e corrigir os erros existentes no *software*, antes deste ser enviado para as equipas de validação. Para além disso, também vão ser avaliadas as melhores ferramentas a utilizar, para a realização dos testes automatizados, uma vez que cada projeto, contém necessidades diferenciadas, e nem todas as ferramentas se enquadram. Em suma, neste estudo foi desenvolvido um projeto de automação num âmbito real escolhendo a melhor ferramenta de automação das selecionadas *Katalon Studio, Robot Framework, Protractor, Watir*, tendo em conta as necessidades da empresa.

ABSTRACT

Nowadays, there are many tasks in which software (SW) plays a critical role, and this will continue to be the case in the future, given the exponential use of Information Technology (IT).

SW's excellence defines the user's loyalty. The fewer flaws the SW has, the greater the probability of achieving user loyalty. To make a good SW's testing is crucial at the development stage since its primary purpose is to identify errors.

Currently, companies or individuals developing SW are already facing a problem: the increased complexity resulting from adding features exponentially. This leads to an impossibility to test the entire SW, every time it is subject to some change. These changes can cause failures in elements that already existed some time ago and were established as stable features. For this reason, only new features and changes were tested, to try to solve this problem, a right approach would be to invest in the implementation of automated tests, without also forgetting the importance and needs of manual tests.

These tests can be implemented to validate the application as a whole. Therefore, core tests will have to be defined to detect failures when there is some change to the SW. In this way, it is guaranteed that a feature, which has not been changed, continues to behave as it should.

In the early days of the development of test automation, two questions were asked: "How are we going to perform these tests?"; "Is the human factor needed to perform this operation? Now, these questions have already been solved. Nowadays, a human does not need to perform the automated tests that have been developed, and the word "automated" in itself means that such tests are performed without human intervention.

This work is inserted in the final project scope of the Master in Information Systems and Technologies for Organizations. It aims to perform an analysis on automated and manual tests, and the benefits of automation in SW development, allowing to select which tools are best suited in a real case.

Several tools have been validated to investigate if they are useful in finding errors. The project was developed at the SIBS company, as part of the IPS - Instant Payments Solution team.

In practical terms, this project is focused on the implementation of a battery of automated tests to find SW failures. This way, it is possible to validate and correct existing errors in the software before it is sent to the validation teams. Also, the best tools to be used to perform the automated tests will be evaluated, since each project contains different needs, and not all tools fit together. In short, in this study, an automation project was developed in a real scope choosing the best automation tool (Katalon Studio, Robot Framework, Protractor, Watir) and taking into account the needs of the company.

PALAVRAS CHAVE

Testes de Software
Automação de Testes
Testes Manuais
Qualidade de Software
Ferramentas de automação de testes
Jenkins
Python
Robot Framework

KEY WORDS

Software Testing
Test Automation
Manual Tests
Software quality
Test automation tools
Robot framework
Jenkins
Python

AGRADECIMENTOS

O desenvolvimento deste projeto advém de um conjunto de esforços que envolvem distintas pessoas, agradecendo:

- Aos meus orientadores, Professor Doutor Filipe Sá e Mestre Anselmo Silva, que estiveram sempre disponíveis para esclarecimento de dúvidas e atentos ao trabalho desenvolvido.

- Aos meus Pais e irmã, pelo percurso académico que me conseguiram proporcionar e pelo apoio e amor que sempre me deram, um muito obrigado!

- Ao António João, pelo apoio, compreensão e motivação durante este longo percurso. Obrigado por todo o amor e dedicação.

- À minha equipa de trabalho, André Reis, Renato Santos, Miguel Almeida, David Silva, Solange Paz, Rui Duarte, Rui Coimbras, Bruno Monteiro, Bernardo Baptista pela partilha de conhecimentos, disponibilidade e companheirismo.

ÍNDICE GERAL

ÍNDICE DE FIGURAS

ÍNDICE DE QUADROS

ABREVIATURAS E SIGLAS

ACK	Acknowledge
ATM	Automated teller machine
CSV	Comma-separated values
DB2	Database 2
FTM	Financial Transaction Manager
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IPS	Instant Payments Solution
JAR	Java Archive
JMS	Java Message Service
KW	Keyword
MQ	WebSphere Message Queue
SW	Software
TI	Tecnologias da informação
TXT	Texto message
XML	Extensible Markup Language

1. Introdução

Neste capítulo serão abordados os objetivos e as motivações para o desenvolvimento deste projeto, bem como uma descrição das metodologias utilizadas para fundamentação do mesmo. Com este estudo pretende-se apresentar uma implementação real, sustentada numa criteriosa revisão da literatura, sobre a automação de testes, de modo a testar e aumentar a confiabilidade do software.

1.1 Motivação

Este projeto foi desenvolvido no âmbito do Mestrado em Sistemas e Tecnologias de Informação para as Organizações, tendo como foco principal a automação de testes.

A integração dos testes é essencial no ciclo de vida de desenvolvimento de software, pois são os testes que viabilizam a identificação de possíveis falhas. Todos os processos de software deveriam conter testes manuais e automatizados, de modo, a ser possível corrigir falhas que apenas são detetadas com os testes, para ser garantida a qualidade de software na entrega ao cliente.

Este tema foi escolhido para o projeto, uma vez, que é bastante satisfatória a entrega de software com a menor quantidade de erros possível. Além disso, é um desafio enorme a análise das aplicações inseridas em projetos, com o objetivo de elaborar casos de testes, que englobem os processos chave para que seja possível, testar o software de forma global. Este é um tema motivante, pois está inserido na atividade profissional do mestrando, que trabalha

numa equipa que pretende diariamente atenuar as falhas existentes no *software* que desenvolve na empresa SIBS. Esta empresa tem mais de 300 milhões de utilizadores e disponibiliza serviços financeiros, modernos, fiáveis e seguros, na área de pagamentos. É responsável pela gestão de redes ATM Express, Multibanco, caixas automáticas, terminais de pagamento automático e meios online ("SIBS,"). O IPS (Instant Payments Solution), está integrado na SIBS, com responsabilidade pelo tratamento das transações instantâneas.

A automação de testes, é algo relativamente recente e tende a agilizar alguns processos repetitivos, com a finalidade de testar todas as funcionalidades de um software, e minimizar os trabalhos repetitivos que são realizados manualmente.

Assim, serão estudadas as ferramentas de automação de testes que melhor se adequam à vertente empresarial, ressalvando-se a necessidade de automatizar, bem como os benefícios da automação.

Dado a importância dos testes manuais e automatizados sobre a qualidade do software, será abordada a relação entre estes dois tipos de testes, assim como a forma como estes se complementam.

1.2 Questões de Investigação e Objetivos

Neste subcapítulo são identificadas as questões que levaram aos objetivos e investigações deste projeto. Além, das questões de investigação, serão apresentados os objetivos gerais, que revelam a ideia central do projeto. Também são abordados os objetivos específicos, que transmitem o que é realizado para os objetivos genéricos serem atingidos.

1.2.1 Questões de Investigação

Com a identificação de questões de investigação, é possível definir de forma clara os objetivos, levando a uma organização das práticas de investigação que melhor se adequam face às problemáticas identificadas, sendo elas:

- Questão 1: Quais ferramentas de automação de testes existem?
- Questão 2: Qual a melhor ferramenta que se enquadra com a estrutura dos desenvolvimentos no contexto onde foi desenvolvida esta investigação, o IPS (*Instant Payments Solution*).

- Questão 3: A automação de testes contribuiu para o aumento da qualidade de software desenvolvido no IPS?

1.2.2 Objetivos Gerais

Um software, em regra geral, inclui muitas funcionalidades e o ser humano não consegue testar todas elas. De modo a evitar um aumento de custos e otimizar o tempo da equipa de testes, considera-se uma boa solução adicionar a automação sobre os processos estáveis e repetitivos.

Executando um conjunto de testes automatizados, para além de, existir um *feedback* constante, relativamente ao estado do software/funcionalidades, as falhas são rapidamente detetadas, sendo possível, efetuar correções antes da entrega do software. Com a união dos testes manuais e automatizados é possível testar mais funcionalidades, garantindo assim, uma maior qualidade no software, que posteriormente será entregue ao cliente.

Este projeto tem como objetivo, adicionar a automação de testes num contexto empresarial, de modo a encontrar erros no SW antes da entrega ao cliente. Para isso foi necessário, avaliar o desempenho de algumas das ferramentas de automação de testes, na minimização dos erros encontrados no processo de desenvolvimento de software, escolhendo a qual melhor se enquadra no âmbito do IPS.

1.2.3 Objetivos Específicos

Para que seja possível concretizar o objetivo geral deste projeto, é essencial planear de forma organizada cada passo a desenvolver/analisar neste estudo. Para isso será adicionado ao projeto uma metodologia ágil denominada de *scrum*.

Adotando este método existirá uma melhor organização e controlo sobre o projeto, uma vez que será possível realizar um acompanhamento pormenorizado sobre as tarefas que estão a ser elaboradas, as que estão planeadas e bloqueios que poderão existir. A metodologia *scrum* foi a escolhida para adaptar a este projeto uma vez que é a utilizada na equipa onde se insere o mesmo.

Foi efetuado um estudo sobre as ferramentas de automação existentes no mercado, que permitiu selecionar uma ferramenta direcionada para a realidade da empresa onde de desenvolveu o projeto.

Assim, foi realizada uma análise dos testes manuais (todos os processos repetitivos e testes estáveis) que já existentes para que, estrategicamente, sejam adicionados aos testes automatizados.

Ao realizar uma análise ao paradigma atual de testes, será possível perceber o que é exequível automatizar aumentando assim as execuções de testes num determinado SW. Deste modo, o leque de funcionalidades a serem testadas será maior, facultando a entrega do software com menos erros ao cliente.

Adicionando o paradigma da automação de testes num ambiente prático, constata-se a importância da criação de baterias de testes de regressão. Estas baterias de testes são constituídas por diversos casos de testes que contemplam os processos de negócio, sendo executadas frequentemente de forma automática.

Com a execução dos testes, garante-se que serão identificados erros mais cedo, existindo uma correção antes do software ser enviado para outras equipas de validação, permitindo assim o envio do software com uma maior qualidade.

A linguagem utilizada será simples, ou seja, o teste será realizado numa linguagem que todas as pessoas consigam perceber (*Keyword-driven*). Com esta solução será relativamente fácil a manutenção/criação de testes. Para além da simplicidade da linguagem, a solução será integrada num repositório para que todos os intervenientes tenham acesso.

A solidez da solução deverá ser mantida através de reutilização de código, boas práticas de programação e manutenção do servidor de execução de testes.

Em termos de enquadramento com a *SIBS*, este projeto tem como objetivo principal, minimizar os erros do software que poderão existir, no software desenvolvido pela equipa do IPS.

Em suma, são identificados os seguintes objetivos específicos:

- Selecionar uma metodologia de desenvolvimento
- Revisão de literatura sobre diversos tipos de testes (Manuais e Automatizados)
- Revisão de literatura sobre ferramentas ligadas a automação de testes
- Análise de processos estáveis e repetitivos na equipa do IPS
- Análise de processos viáveis para automatizar
- Criação da especificação de testes
- Criação de casos de testes mais repositório (Git)
- Criação de baterias de execução de testes
- Avaliação de melhorias de qualidade do SW

1.3 Metodologia de Investigação

Este projeto vai levar a cabo, numa etapa inicial, uma criteriosa revisão sistemática da literatura, de forma a contextualizar e verificar quais os testes de software existentes e quais as ferramentas mais adequadas e utilizadas para esse fim.

Segundo (Sampaio & Mancini, 2007) a revisão sistemática é considerada uma pesquisa que utiliza como fonte a literatura tendo em conta um determinado tema. Nas revisões sistemáticas são utilizados os seguintes métodos: levantamento das questões de investigação; pesquisa da literatura; seleção dos artigos; extração dos dados; avaliação da qualidade metodológica; síntese dos dados (metanálise); avaliação da qualidade das evidências; redação e publicação dos resultados (Pereira & Galvão, 2014).

Uma revisão de literatura deverá conter certas características, tais como, analisar e sintetizar uma literatura de forma a sustentar um tópico de investigação, é fornecida uma base para um tópico de investigação, facultar uma base firme à seleção de metodologias de investigação e demonstrar que a investigação proposta contribui com algo novo para o corpo geral de conhecimento ou para a melhoria de conhecimento na área (Levy & Ellis, 2006).

Partindo dos pensamentos referidos, foram definidos os critérios de inclusão e exclusão para seleção da bibliografia. Referente ao tema foram identificados os seguintes critérios.

1. Artigos publicados a partir de 2002
2. Existe referencia com data 1970, pois é bastante referenciada;
3. Artigos escritos em língua inglesa, portuguesa e espanhola;
4. Artigos na temática de testes de software;
5. Artigos com evidência científica, quantitativos ou qualitativos.

Todas as pesquisas bibliográficas efetuadas, têm como tema central automação de testes e foram utilizadas as seguintes palavras-chave: Testes de software, testes manuais de software, testes automatizados de software, qualidade de software, ferramentas de automação de testes.

Estas pesquisas foram efetuadas a partir de bases de dados científicas, como por exemplo ScienceDirect, SpringerLink, IEEE Xplore, ACM e B-on.

1.4 Cronograma

Com o objetivo de planear e organizar as tarefas foi elaborado um cronograma. Estas tarefas são planeadas para que seja possível a realização destas num determinado tempo, de modo a não comprometer a data de finalização do projeto.

Na Figura 1 podemos ver a organização inicial do cronograma, onde representam os meses e as tarefas realizadas. A revisão da literatura e estado da arte, são tarefas que acompanham todas as fases do projeto, devido a isso foi planeada desde janeiro até outubro. Em fevereiro e março, o plano recai sobre a análise de requisitos e caso de testes, direcionando também à especificação dos mesmos. Após a especificação de testes estar concluída, planeou-se a fase de instalação e desenvolvimento de scripts nos meses de abril, maio, junho e julho. Nos meses de julho até setembro a planificação direcionou-se para a instalação e configuração do *jenkins*, criando assim as baterias de testes. A finalização e revisão do documento foi projetada para os meses de novembro e dezembro.

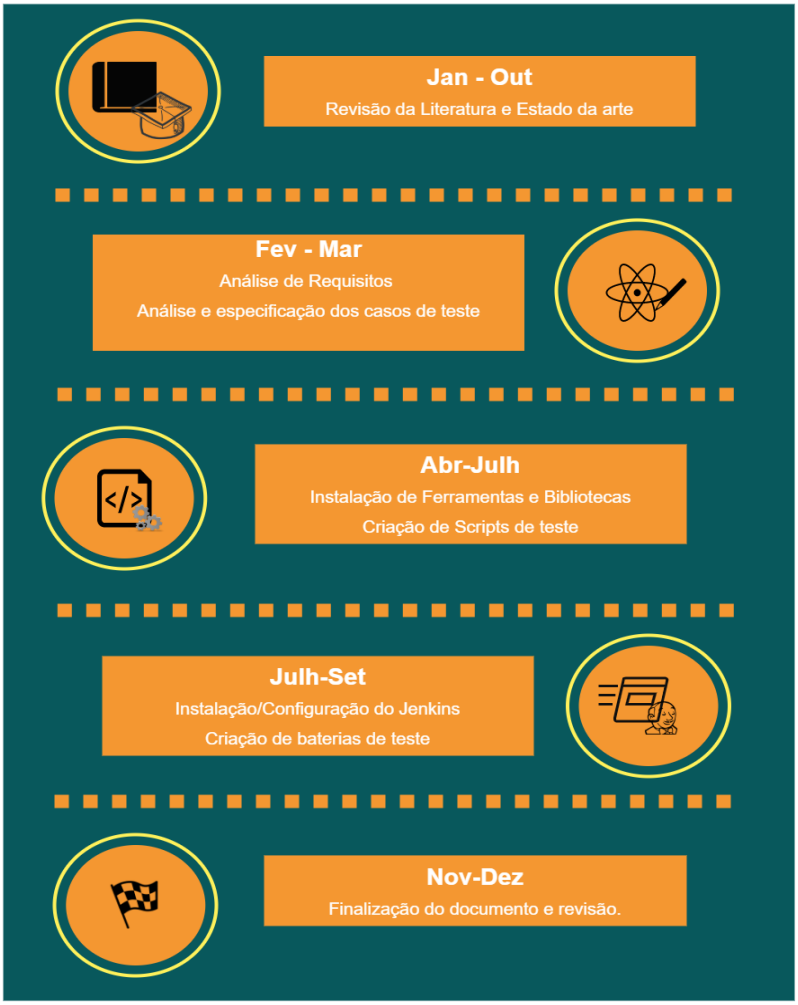


Figura 1 – Cronograma Inicial

O planeamento inicial Figura 1 sofreu alterações à medida que foi iniciado o projeto. Existiu um atraso na disponibilidade para reunião de análise dos casos de teste importantes a desenvolver numa fase inicial. De igual modo, existiram atrasos na instalação das ferramentas, uma vez, que as máquinas de trabalho não têm acesso à internet, logo todas as bibliotecas e dependências foram instaladas de forma manual. O ANEXO 1 representa de forma detalhada, o cronograma realizado neste projeto.

1.5 Estrutura da Tese

Este documento, encontra-se dividido em quatro capítulos.

No primeiro capítulo encontramos a introdução, onde são apresentados os objetivos, a motivação para a escolha do tema do projeto, as questões e metodologias de investigação que são importantes para organizar as práticas de investigação com o objetivo de responder a todos os objetivos pretendidos.

No segundo capítulo, é apresentado o estado da arte sobre conceitos ligados ao tópico deste projeto. São abordados vários temas com base científica, sobre testes de software, definição de qualidade, objetivos dos testes, os tipos de testes de software, testes manuais e automatizados. Dentro dos testes automatizados, apresenta-se uma justificação sobre a necessidade de automatizar, assim como, os tipos de testes automatizados e os benefícios destes, sobre a vertente empresarial. No que toca a ferramentas automação de testes de software, são descritas quatro das disponíveis no mercado para análise de modo a ser possível selecionar a ferramenta que melhor se enquadra no caso de estudo.

No terceiro capítulo, é apresentado a parte prática deste projeto realizado na SIBS. São abordados de forma detalhada o problema, as etapas realizadas e as diversas soluções para o mesmo. São esclarecidos os tipos de testes realizados e a sua especificação, as ferramentas utilizadas para o desenvolvimento dos casos de teste e as ferramentas usadas para a execução dos mesmos. Os benefícios da integração da automação de testes na equipa do IPS também são referenciados.

As conclusões finais do projeto e as tarefas a realizar, num futuro próximo são identificadas no quarto e último capítulo.

No anexo 1, pode ser consultado o cronograma real de forma detalhada, das etapas deste projeto. Os procedimentos de instalação do *jenkins* estão visíveis no anexo 2 e o anexo 3 apresenta a especificação de todos os casos de testes desenvolvidos

2. Revisão de Literatura

Neste capítulo será apresentado o estado da arte onde são abordados conceitos sobre definição de qualidade, testes de software, que abrange os objetivos destes e os tipos existentes, manuais e automatizados. Dentro do tema da automação é fundamentada a necessidade de automatizar, assim como os tipos de testes automatizados e os benefícios dos mesmos, numa vertente empresarial.

No que diz respeito às ferramentas de testes de software, serão descritas quais é que existem, e quais serão utilizadas.

2.1 Porque é necessário testar?

Neste momento, e em várias ações da nossa vida quotidiana, estamos dependentes de diversos softwares e da sua execução de forma correta, tanto nos nossos equipamentos eletrónicos como em ações que fazemos e é executado software de externos, como por exemplo realizar uma transferência bancária ou fazer compras online (Homès, 2013).

Segundo a formação ISTQB (2011) um software que não funciona como é esperado, pode causar diversos problemas, como perda de dinheiro, tempo ou reputação comercial.

Erros humanos, podem estar presentes em qualquer fase do ciclo de desenvolvimento de software, e dependente do tipo de erro cometido, o impacto pode ser banal ou catastrófico, dependendo das suas consequências (Graham, Van Veenendaal, & Evans, 2008; Homès, 2013; Sharma).

Defeitos, erros e falhas são conceitos um pouco confusos, uma vez que parecem ter todos o mesmo significado, o que não é uma suposição correta. Quando se fala em falhas, deve-se associar à incapacidade de um sistema/componente executar funções necessárias para determinados requisitos. Klaus Olsen (chair) (2018) refere que o defeito está associado a um processo ou definição de dados incorretos, dentro de um software ou documento. As ações incorretas provocadas pelo ser humano denominam-se de erro (Klaus Olsen (chair), 2018; Lazic, 2020).

A Figura 2 representa uma explicação sobre o processo que origina falhas no software. Um ser humano comete um **erro**, que origina um **defeito** que produz uma **falha** no sistema/software (Klaus Olsen (chair), 2018). Por exemplo, a pessoa que está a desenvolver o software realiza uma operação com uma variável que não foi devidamente declarada e envia o software sem a realização de testes. Este erro humano provocou um defeito no sistema, que neste momento se encontra disfarçado, pois, não existiu nenhuma execução do software. Quando o software é executado e/ou utilizado por alguém, vai originar uma falha devido ao erro provocado pela pessoa que estava a desenvolver o software.

Figura 2 - Processo de falha SO e SW (Klaus Olsen (chair), 2018)

Na visão de Ko and Myers (2005) os defeitos, são a maior ameaça no que diz respeito à confiabilidade do software, sendo que quem desenvolve software despende em média 70-80% do tempo em testes.

Graham et al. (2008) e Uddin and Anand (2019), mencionam que é necessário testar porque todos cometemos erros e as suas consequências podem ser bastante devastadoras, necessitamos de verificar tudo o que produzimos.

Segundo a formação (ISTQB, 2011), existem razões reconhecidas para justificar o porquê de ser necessário testar, sendo estas:

- Para encontrar defeitos
- Para reduzir o risco
- Para garantir que o software realiza os requisitos
- Para dar confiança
- Para alcançar conformidade
- Para medir a qualidade

2.2 Testes de Software

Na visão de Hooda and Chhillar (2015), Lewis (2017) e Rierson (2017) os testes de software numa visão geral são conhecidos como atividades para encontrar erros ou processos que analisam um item de software tendo como objetivo encontrar diferenças entre as condições existentes e os requisitos necessários garantindo a qualidade de software (Agarwal, Tayal, & Gupta, 2010). Essa qualificação está associada a uma medição dos atributos face às expectativas e padrões aplicáveis (Agarwal et al., 2010).

Segundo Kan (2002) o tema de qualificação de qualidade é considerado um termo ambíguo dependendo do interesse e atributos de qualidade das entidades envolvidas. O termo qualidade faz parte do nosso quotidiano e tem significados diferentes em usos populares e profissionais (Kan, 2002). Numa visão popular a qualidade pode ser sentida e julgada mas em contrapartida não pode ser medida, isto porque cada pessoa tem a sua própria perceção e interpretação da mesma (Kan, 2002). Numa visão profissional o conceito qualidade tem de ser determinado, para que seja possível existir uma definição abrangente para toda vertente profissional. Visto que existia essa necessidade, Juran e Gryna (1970) e de Mello Cordeiro (2004), definiram a qualidade como uma adequação de uso, ou seja, é necessário considerar a expectativa e requisitos dos clientes uma vez que são os mesmos que vão usar o produto e este deve estar preparado para as suas necessidades. Para que exista uma adequação de uso, todos os clientes devem usar o produto mesmo tendo maneiras diferentes de o fazer. Em 1979 foi reforçada essa definição acrescentando que a qualidade é estar em conformidade com requisitos Crosby (1979), o que vai implicar uma clareza na sua especificação de modo a que estes sejam bem interpretados.

No ponto de vista de Ammann and Offutt (2016) os testes contemplam atividades padrão bastante importantes onde devem ser criados os requisitos de testes, que servem de base para a criação de valores reais e scripts. Estes scripts são convertidos em testes executáveis sendo

analisado o output destas execuções, e com esta análise, teremos o estado do teste determinado, revelando assim um sucesso ou falha. Com a análise de cobertura de critérios de testes conseguem-se decidir os melhores parâmetros de entrada na execução do teste e com isto aumentar a probabilidade de encontrar problemas, tornando o software mais confiável. (Ammann & Offutt, 2016)

Várias pessoas calculam que um teste de sucesso é aquele que não encontra erros, mas numa visão de teste esta ideia não está correta. Um teste bem-sucedido é aquele que encontra erros (Rierson, 2017).

Neste processo tanto os componentes como os requisitos são avaliados manualmente, ou recorrendo a ferramentas de automação confirmando que o comportamento está enquadrado com os requisitos pretendidos (Hooda & Chhillar, 2015).

2.2.1 Ciclo de vida dos testes

Segundo Prof . Swati Dubey (2017) os testes são considerados a fase mais crítica do ciclo de vida do desenvolvimento de software, uma vez que definem as várias atividades que devem ser executadas constantemente e em sequência, com o objetivo de avaliar o mesmo (Hooda & Chhillar, 2015; Swati, 2020; Testim).

A empresa Testim referencia que podem estar numa organização os melhores programadores, mas mesmo sendo os melhores podem cometer erros. Com a realização das etapas do ciclo de vida dos testes, é possível encontrar esses erros e corrigir os mesmos, para além disso também é uma forma de economizar tempo e esforço (Afzal, 2007).

Cada etapa, contém diferentes objetivos e resultados, sendo cada etapa crucial no teste de um determinado produto (Swati, 2020; Testim), existindo um trabalho coordenado entre a equipa de testes com a equipa de desenvolvimento (Testim). O ciclo de vida integra seis fases (Figura 3), sendo elas:

Análise de requisitos: Durante esta fase serão analisados os requisitos funcionais, onde é definido o que o software deve fazer, e não funcionais, que determinam a segurança e desempenho do sistema (Hooda & Chhillar, 2015; Prof . Swati Dubey, 2017; Testim). Para além dos requisitos, nesta fase são criados documentos de especificação funcional e técnica (Hooda & Chhillar, 2015; Swati, 2020). Segundo Swati (2020), assim, a equipa consegue adquirir conhecimento detalhado dos testes a realizar, tendo noção das prioridades dos mesmos. Caso alguns requisitos sejam impossíveis de testar, devido a limitações de sistema e

ambiente de testes , devem ser comunicados à equipa de negócio, para que a estratégia de mitigação seja planeada (Hooda & Chhillar, 2015; Swati, 2020).

Fase de planeamento: na visão de Afzal (2007) esta etapa é uma das chaves para o sucesso dos testes.

Nesta fase existe uma descrição do contexto e dos objetivos depois de ser compreendido o produto, a equipa, vai analisar os riscos e define cronogramas e ambientes de testes de modo a ser criada uma estratégia. Depois disto, quem está a gerir atribui funções e responsabilidades às restantes pessoas tendo em conta as suas habilidades (Testim).

São consideradas as questões de estratégia de testes, recursos, responsabilidades, riscos e prioridades, com o objetivo de serem garantidas as metas pretendidas (Afzal, 2007; Swati, 2020; Testim).

De forma estruturada, é criado o plano de testes, descrevendo os tópicos descritos anteriormente (Afzal, 2007; Graham et al., 2008; Hooda & Chhillar, 2015; Prof . Swati Dubey, 2017).

Design: Nesta fase, os objetivos gerais do teste são transformados em condições tangíveis de teste (Graham et al., 2008; Swati, 2020). São identificados os inputs para iniciar o design do caso de teste, a criação de cenários e os dados. Nesta fase é documentado, de forma detalhada, como os procedimentos conducentes à realização do teste (Afzal, 2007). Caso esteja a ser desenvolvida a automação de testes, são igualmente criados os scripts. Antes de entrar na fase de Implementação, devem ser originados casos de teste, dados e scripts de automação (Prof . Swati Dubey, 2017; Testim).

Em suma, a fase de design consubstancia-se através das seguintes tarefas (Swati, 2020):

- Detalhar a condição de teste, dividindo-as em várias subcondições de modo a aumentar a cobertura dos testes.
- Identificar e obter dados de teste
- Identificar e configurar o ambiente de testes
- Criação de métricas de para acompanhar possíveis mudanças de requisitos
- Criação de métricas de cobertura de teste

Implementação e execução: Na fase de implementação, Graham et al. (2008) afirmam que se reúne um design de alto nível e começa-se a construir e a implementar os testes, podendo ou não usar ferramentas de automação (Graham et al., 2008; Prof . Swati Dubey, 2017). Os ambientes de teste são configurados, com os requisitos necessários para a execução. Após implementação e configuração de sistema, procede-se à execução dos testes (Prof . Swati Dubey, 2017; Testim). Nesta fase, os testes são executados de forma manual ou automatizada. Após execução, os resultados são registados, comparando assim o *output* do teste com o cenário real, validando que o *output* da execução era o esperado. Os erros encontrados nas execuções são enviados para a equipa de desenvolvimento, com o intuito de serem corrigidos. Após efetuadas as devidas alterações, serão executados novamente os testes, para validar se o defeito foi emendado (teste de confirmação), garantindo que não foram acrescentados outros defeitos com as alterações acrescidas (Graham et al., 2008).

Encerramento do ciclo de testes: Nesta fase os dados das atividades de testes são recolhidos, com o objetivo de solidificar experiências, tendências, factos e números (Graham et al., 2008; Prof . Swati Dubey, 2017).

Os testes podem cessar por diversos motivos, sendo eles: teste realizado com sucesso; informação necessária para o teste não está bem especificada; o projeto foi cancelado; os objetivos foram todos atingidos (Graham et al., 2008).

A fase do encerramento de ciclo de teste compreende as seguintes tarefas:

- Validar que todas as tarefas pretendidas foram entregues;
- Criar relatórios de resumo do teste e sua execução;
- Entrega dos testes para a equipa de manutenção (Prof . Swati Dubey, 2017; Swati, 2020; Testim).

Esta fase é bastante construtiva, uma vez que fornece à equipa de testes um grande detalhe de todo o processo que foi realizado (Testim). Com este detalhe, a equipa tem material necessário para reunir, discutir e analisar o que correu bem ou mal. Assim, futuramente, têm conhecimento para adotar novas estratégias, obtendo um maior sucesso nos projetos seguintes (Prof . Swati Dubey, 2017).

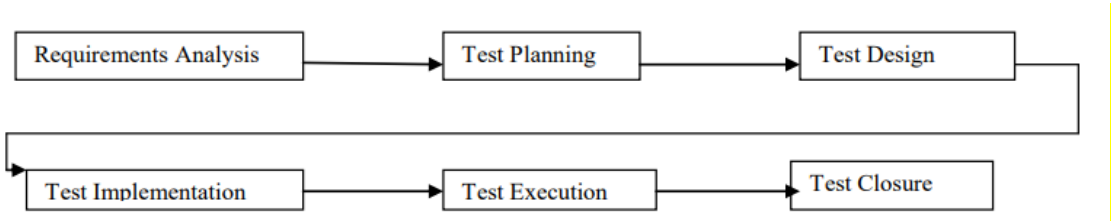


Figura 3- Ciclo de vida dos testes (Prof . Swati Dubey, 2017)

2.2.2 Níveis de testes de Software

Os testes podem ser projetados e construídos durante todas as etapas de desenvolvimento do software, sendo integrados em cada fase, e não apenas na etapa final de desenvolvimento (ISTQB, 2011). As partes mais demoradas do teste são, na verdade, o design e a construção do mesmo, portanto, as atividades de teste podem e devem ser realizadas durante todo o desenvolvimento.

Cada nível de teste (**Testes de aceitação, testes de sistema, testes de integração, e testes unitários**) acompanham cada atividade de desenvolvimento de software (Ammann & Offutt, 2016). O modelo V desenvolvido por CRAIG Craig and Jaskiel (2002), representado na Figura 4, expressa este acompanhamento entre a fase de desenvolvimento e de testes. Dependendo do projeto e do tipo de produto de software, o modelo V pode conter mais ou menos níveis de testes (ISTQB, 2011).

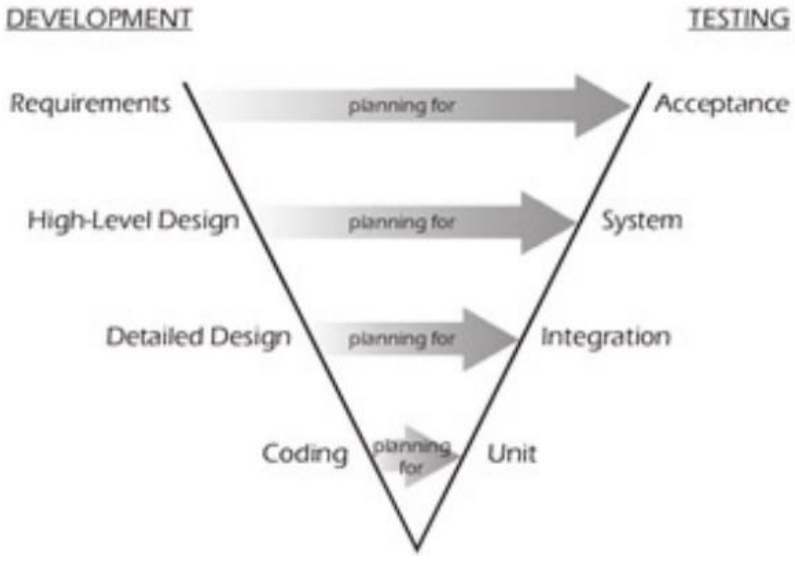


Figura 4 – Modelo V (Craig & Jaskiel, 2002)

Os testes são compostos por quatro níveis: Testes de aceitação; Testes de sistema; Testes de integração; Testes unitários.

Lewis (2017) e Agarwal et al. (2010) afirmam que o**s testes de aceitação** são feitos pelos clientes de modo a validar se o software está em conformidade com os requisitos indicados, sendo que nesta fase o cliente irá decidir se aceita ou rejeita a entrega do *software*. Estes testes podem ser realizados por longos períodos de tempo, com a finalidade de descobrir erros aglomerados que possam danificar o sistema ao longo do tempo (Agarwal et al., 2010).

Os testes de sistema são testados num ambiente operacional antes de serem executados os testes de aceitação (Lewis, 2017). Estes tipos de testes concentram-se em encontrar bugs que possam resultar de interações entre subsistemas e componentes do sistema, verificando também as características não funcionais existentes (Agarwal et al., 2010), incluindo testes de usabilidade, desempenho, compatibilidade e stress.

Os testes de integração servem para descobrir erros associados interface, requerendo-se que todos os módulos sejam testados de forma íntegra, sendo que o teste é criado gradualmente, adicionando-se lentamente os módulos que já foram testados unitariamente (Agarwal et al., 2010). Os testes de integração visam validar a possibilidade de integrar os diferentes módulos, não obtendo erro na passagem de parâmetros, na chamada de cada módulo individual (Lewis, 2017). Deste modo, fomenta a criação de um plano de inclusão que detalha as etapas e as sequências da introdução/execução dos módulos.

No que diz respeito aos **testes unitários,** segundo Agarwal et al. (2010), estes servem para avaliar o software com foco nas implementações realizadas, tendo como base componentes individuais. Segundo Agarwal et al. (2010) e Ammann and Offutt (2016), este nível concentra-se no esforço de verificação de limites, testando cada componente de forma independente. Este nível tem como objetivo validar o código no design de alto e baixo nível, garantindo que todas as ramificações sejam executadas, validando mensagens de erro, códigos e output, confirmando que o código desenvolvido está coerente com o que está especificado nos ficheiros (LLD e HLD) (Agarwal et al., 2010; Ammann & Offutt, 2016; Lewis, 2017).

2.3 Tipos de Testes de Software

Base36 (21 Mar. 2015) referencia a importância de existir uma definição e distinção sobre os testes manuais e automatizados, uma vez que, os defeitos de software são detetados pelo fator humano, mais propriamente pelos testes manuais, complementares à automação de testes.

É bastante difícil encontrar bugs no sistema de forma eficiente, enquanto que a automação está focada na remoção/diminuição de tarefas repetitivas e simples, encontrando falhas neste tipo de processos, os testes manuais encontram a maior parte dos novos bugs que podem aparecer no software (J. Itkonen, 2009).

2.3.1 Testes Manuais

Os testes manuais são entendidos como sendo testes executados manualmente na visão da empresa Base36 (21 Mar. 2015), pelas pessoas que desenvolvem o software ou por pessoas direcionadas para a realização dos mesmos. Este tipo de teste são executados sem uso de ferramentas ou scripts, Bhatt (2017) sendo adequado a projetos menores ou em projetos em que as especificações estão em constante mudança (Base36, 21 Mar. 2015).

Uma vez que, neste tipo de testes o humano tem de medir tempos de resposta e comparar resultados, estes não são uma boa opção, existindo uma maior probabilidade de erro humano, originando uma menor confiabilidade no mesmo (Bhatt, 2017).

No entanto, ao recorrer aos testes manuais, é possível encontrar problemas reais, algo que pode acontecer com o utilizador, algo que o testes automatizados não podem verificar, pois não estão preparados para validar um determinado comportamento (Base36, 21 Mar. 2015). Os testes manuais estão preparados para a mudança, uma vez que é mais rápido mudar o paradigma da execução manual. Por sua vez a mudança na automação de testes carece de alterações de scripts de execução, pelo que seria, mais demorado comparativamente ao teste manual (Base36, 21 Mar. 2015).

2.3.2 Testes Automatizados

Nos testes automatizados são usadas ferramentas e scripts de execução (Base36, 21 Mar. 2015). Estes testes têm como objetivo repetir ações predefinidas comparando os requisitos com o resultado real do teste (Bhatt, 2017). Se estes pontos não coincidirem, será necessário

analisar o código e alterar o mesmo. Os testes automatizados abrangem uma maior cobertura e rapidez na sua execução (Bhatt, 2017). Além disso, é-lhes associados um leque de vantagens

Os testes são executados de forma rápida e eficaz, sendo um processo rentável a longo prazo. Os dados de teste são preenchidos automaticamente o que leva a uma menor exaustão para a realização do mesmo. O output de cada teste pode ser validado por qualquer pessoa, uma vez que está visível a todos, algo que não se verifica nos testes manuais. Em comparação com os testes manuais, o teste automatizado traz mais benefícios em projetos grandes (Mailewa, Herath, & Herath, 2015).

No desenvolvimento deste tipo de testes, segundo Garousi and Elberzhager (2017) existem seis atividades que teremos de ter em consideração.

1. Design de caso de teste;
2. Teste de script;
3. Execução do teste;
4. Teste de avaliação;
5. Relatório de resultados do teste;
6. Gerenciamento de testes e outras atividades de engenharia de teste.

2.3.3 Tipo de teste que se enquadra no caso de estudo

Este caso de estudo, enquadra-se principalmente na automação de testes, tendo foco nas ferramentas de automação e na criação de uma bateria de testes. Não desvalorizando os testes manuais, o principal objetivo recai na criação de testes automatizados, de modo a realizar testes em funcionalidades consideradas estáveis.

A automação de testes traz à equipa do IPS uma maior execução e cobertura, minimizando o tempo da execução de um teste realizado de forma manual. Com este tipo de testes, conseguimos perceber se existiu alguma regressão no software com os novos desenvolvimentos, ou correções necessárias, obtendo-se um feedback continuo e rápido.

Com o projeto desenvolvido, é possível termos uma estrutura de testes que pode ser executada a qualquer momento e de forma simultânea, sendo que este tipo de estrutura ajuda a validar o software. Por exemplo em situações que envolvem uma correção, esta tem impacto em diversas funcionalidades do software e era impraticável realizar testes manuais a todas elas num determinado período de tempo. Por oposição, com a automação conseguimos executar uma maior cobertura de testes, num tempo reduzido.

2.4 Estruturas de automação de testes

As *frameworks* de automação são usadas para a execução de testes com base em algoritmos tendo como objetivo a comparação de resultados reais do script com resultados esperados dos requisitos (Milad Hanna 2018; Vila, Novakova, & Todorova, 2017).

Segundo Milad Hanna (2018), a utilização de *framework* de testes facilita o processo do trabalho, uma vez que permite um desempenho mais rápido em comparação com o ser humano. No ponto de vista de Vila et al. (2017), estas ferramentas são relevantes para a reutilização de testes/scripts. Por exemplo, um caso de teste que necessite de ser executado várias vezes, com o auxilio da *framework* vai ser bastante mais rápido (Vila et al., 2017).

As ferramentas de automação de testes, podem ser divididas em distintas categorias, nomeadamente:

- **Ferramentas de testes unitários**

Um teste unitário valida cada unidade de forma isolada comparando o real com os resultados esperados (Olan, 2003). Com este tipo de testes os erros serão detetados mais cedo impedindo assim a propagação noutras partes do projeto. (Huizinga & Kolawa, 2007)

Estes podem ser gerados de forma automática, tendo auxílio as ferramentas de testes unitários que irão testar o funcionamento correto de um método. Para além da validação de funcionamento do método, verificam a estrutura do código e as boas práticas de programação. (M. A. a. C. Umar, Zhanfang, 2019). Ferramentas com JUnit, NUnit, JMockit simplificam o desenvolvimento dos casos de testes (Olan, 2003; M. A. a. C. Umar, Zhanfang, 2019).

- **Ferramentas de testes funcionais**

Os testes funcionais, para além de avaliarem a qualidade de software, ajudam na qualidade do mesmo. Este tipo de testes são utilizados na especificação de requisitos (Andrea, 2007), de modo a garantir que os mesmos estão em conformidade com os requisitos do utilizador (M. A. a. C. Umar, Zhanfang, 2019).

As ferramentas de testes funcionais avaliam um software tendo em conta os requisitos definidos (M. A. a. C. Umar, Zhanfang, 2019). Estes tipos de testes geralmente estão ligados a automação de testes, regendo-se pela analise do software sobre requisitos funcionais. Este tipo de teste geralmente fazem a simulação do comportamento humano perante o software

(Schwaber & Gilpin, 2005). Ferramentas como Selenium, HP QuickTest Professional são compatíveis com o tipo de testes mencionados.

- **Ferramentas de análise da cobertura de código**

Na visão de Altvater (2017) as ferramentas de análise da cobertura de código, são utilizadas para medir quais as partes do software que estão cobertas por testes automatizados. Com esta medição conseguimos perceber quais são as instruções que estão desprotegidas e irão originar defeitos (Atlassian) . A cobertura do código é medida com o objetivo de saber se os nossos testes estão a testar o código de forma eficaz, se os testes desenvolvidos são suficientes, e se é exequível manter a qualidade do teste durante o ciclo de vida de um projeto (Altvater, 2017). Segundo a empresa Atlassian , esta medida está em valores percentuais, ou seja, quanto mais próximo de 100% existe uma menor probabilidade do software conter erros não detetados pelos testes.

CodeCover, *EMMA*, *PITest*, *Atlassian Clover* são algumas ferramentas associadas aos testes de análise de cobertura de código (M. A. a. C. Umar, Zhanfang, 2019).

- **Ferramentas de gestão de testes**

Este tipo de ferramentas é usado com o objetivo de armazenar informações. Estas informações indicam-nos como o teste deve ser realizado, esboçar o planeamento das atividades, bem como elaborar o relatório do estado das atividades (Barna, Litoiu, & Ghanbari, 2011). Em suma, este tipo de ferramentas otimizam o processo de teste pelo qual permite uma análise de dados de forma mais rápida, dando às equipas informação importante para estas gerirem os projetos de forma mais fácil. Test Manager, Test Link, TETwar, Mantis são algumas das ferramentas de gestão de testes.

- **Ferramentas de performance de testes**

No ponto de vista de Barna et al. (2011) estes tipos de testes são realizados, para avaliar o desempenho de componentes de software e do software como um todo. Com este tipo de testes é possível descobrir problemas funcionais e de desempenho (Barna et al., 2011).

Na visão de M. A. a. C. Umar, Zhanfang (2019) este desempenho é medido por três métricas diferentes:

- Velocidade;
- Escalabilidade;

- Estabilidade.

No que toca à velocidade, é analisado o tempo de resposta do sistema, sendo a carga máxima sobre o sistema medida pela escalabilidade. A estabilidade informa se o software é estável em situações de carga. Os testes de stress e carga estão categorizados como testes de performance (Barna et al., 2011).

Para este tipo de testes são usadas as seguintes ferramentas, JMeter, Rational Performance Tester, HP LoadRunner (Hussain, Wang, Toure, & Diop, 2013; M. A. a. C. Umar, Zhanfang, 2019).

Para além das diferentes categorias das ferramentas de automação de testes, também existem diversos tipos de estruturas associadas às *frameworks* de automação:

- **Estrutura de automação linear**

Neste tipo de estrutura, a pessoa que está a desenvolver o teste não necessita de escrever código, uma vez, que é uma estrutura de gravação e reprodução (Aebersold; De Grood, 2008). Neste tipo de estrutura, é gravada cada etapa como navegação, registo de entrada de utilizadores e os pontos onde existe a necessidade de verificações no teste, após esta gravação o script é reproduzido de forma automática (Aebersold; De Grood, 2008).

Na visão de Aebersold e De Grood (2008), não é necessário ter experiência em automação para serem desenvolvidos os testes pois os scripts são bastante rápidos de criar uma vez que estes vão ser gerados a partir de gravações. Aebersold complementa que é mais fácil a compreensão do teste, dado que os scripts são organizados de forma sequencial, sendo que com as gravações de ações não é necessário realizar planeamentos de elaboração de scripts.

Apesar das vantagens mencionadas anteriormente, este tipo de estruturas contém diversas desvantagens. Neste tipo de estrutura os scripts desenvolvidos não são reutilizáveis, os dados de entrada e/ou validações são codificados no script de gravação, o que leva a uma limitação, quando se pretende executar o script de teste, com um conjunto de dados diferentes (Aebersold; De Grood, 2008). Em termos de manutenção também é mais trabalhoso, uma vez que terá de efetuar novas gravações dos steps que sofreram as alterações (Aebersold).

- **Estrutura modular**

No que toca à estrutura modular, segundo Bajaj (2015) existe um maior controlo sobre o fluxo dos scripts, uma vez que, os testes são divididos em unidades, funções e secções separadas, com o objetivo das mesmas serem testadas de forma isolada (Aebersold; Bajaj, 2015). Com a separação de módulos, é possível construir testes maiores de forma hierárquica, o que irá representar vários casos de testes (Aebersold).

Caso exista alguma alteração na aplicação que está a ser testada, com este tipo de estrutura só sofre alterações o módulo do script de teste que está direcionado à aquela fase do teste e, consequentemente, é mais fácil e rápido a correção de possíveis alterações nas aplicações a serem testadas (Aebersold). Este tipo de estrutura também contém outra vantagem no que toca a reutilização, uma vez que, os módulos/funções podem ser utilizados em diversos casos de testes (Aebersold; Bajaj, 2015).

Neste tipo de estrutura, estão identificadas as desvantagens de os dados serem codificados, logo não é possível a utilização de diferentes dados (Aebersold; Bajaj, 2015). Para além disso, nesta estrutura é requerido o conhecimento de programação (Aebersold).

- **Estrutura da arquitetura da biblioteca**

Segundo Bellatrix (2019) e Aebersold esta estrutura, é baseada na estrutura modular. Em vez do teste conter vários scripts que necessitam de execução, as tarefas semelhantes que se encontram dentro dos scripts são agrupadas por funções para que seja possível realizar a chamada das mesmas pelos scripts de teste sempre que seja essencial (Aebersold; Bellatrix, 2019). Estas funções são mantidas em bibliotecas, e este tipo de estrutura torna a manutenção e escalabilidade do teste mais fácil e económica, pois, existe um alto nível de modularização (Aebersold). Devido à biblioteca de funções comuns que existe nesta estrutura, subsiste um grande grau e reutilização, sendo possível usar os módulos de vários scripts de testes (Aebersold). Em termos de desvantagens, os dados de testes ainda se encontram no script, ou seja, se for necessário alterar algum dado de teste é necessário alterar o script, e os scripts de teste demoram mais tempo a serem desenvolvidos (Aebersold).

- **Estrutura baseada em dados**

Neste tipo de estrutura existem scripts de testes parametrizáveis, ou seja, existe uma separação dos dados de teste e do script principal (Aebersold; De Grood, 2008). Nesta

estrutura é possível executar scripts com dados diferentes, levando ao reaproveitamento do script para diversos casos que necessitam de ser testados, o que leva a uma redução de tempo em termos de novos desenvolvimentos (Aebersold; Bajaj, 2015).

Estes dados provêm de dados inseridos em excel, .txt, csv, Base de dados (Aebersold; Bajaj, 2015). Os scripts possuem uma maior capacidade no que diz respeito à manutenção, tendo também uma maior cobertura os testes automatizados, devido à vantagem de ser possível executar o mesmo script com diferentes dados de input/output (De Grood, 2008).

Não obstante, esta estrutura, contém desvantagens, uma vez que é necessário contratar uma pessoa especializada com conhecimento de diversas linguagens de programação e de modo a configurar a estrutura baseada em dados e a organização das fontes externas, assegurando a conexão destas com os scripts principais (Aebersold; Bajaj, 2015).

- **Estrutura baseada em palavras-chave**

A estrutura baseada em palavras-chave requer um desenvolvimento das palavras-chave de forma independente. Estas palavras-chave representam uma serie de ações, e a sua ordem de execução (Aebersold; Bajaj, 2015). Nesta abordagem também é possível realizar a separação dos dados com o script em si, sendo cada palavra-chave documentada de forma a ser percetível os passos produzidos pela mesma (Aebersold; Bajaj, 2015).

Esta estrutura contém diversas vantagens, não sendo necessário um conhecimento profundo de *scripting*, as palavra-chave criadas podem ser utilizadas em todos os scripts que pretenderem, e os scripts de teste podem ser criados de forma independente da aplicação de teste (Aebersold; Bajaj, 2015). Por outro lado, é necessário alto conhecimento de programação para a criação da estrutura baseada em palavra-chave, sendo o investimento inicial bastante alto, podendo as palavras-chave revelarem-se um incomodo de manter ao estimar uma operação de teste (Aebersold; Bajaj, 2015).

- **Estrutura de teste híbrido**

Este tipo de estrutura, advém de uma combinação de qualquer uma das estruturas mencionadas, adequando-as às necessidades da automação. Possui a vantagem de aproveitar os benefícios das outras estruturas e aliviar as suas fraquezas (Aebersold; Bajaj, 2015).

2.5 Ferramentas de Automação de Testes

Existem diversas ferramentas de automação disponíveis no mercado. Neste capítulo são identificadas quatro delas, referindo-se as suas vantagens e desvantagens. No final deste capítulo existe uma comparação das quatro ferramentas, justificando o porquê de ser usada a *robot framework* no projeto. Estas quatro ferramentas foram selecionadas a partir de uma lista de ferramentas fornecidas pela empresa.

Neste sentido, o *katalon studio* foi selecionado, devido às suas funcionalidades de gravação e análise, no que toca a *robot framework*, destacou-se por estar associada ao *Python*[1], e a utilização de *keyword-driven*. *Protractor* estar associado a casos de teste em angular, algo utilizado na equipa onde foi realizado o projeto. O *Watir* utiliza *ruby*[2], sendo uma linguagem que contém diversos recursos, realçando esta ferramenta para integração no caso de estudo.

2.5.1 Katalon Studio

Katalon studio, é uma plataforma associada aos testes automatizados que contém vários recursos para que seja possível automatizar testes para aplicações web, aplicações móveis e API (M. A. Umar & Zhanfang, 2019).

Segundo a visão da empresa katalon e de M. A. a. C. Umar, Zhanfang (2019), esta aplicação foi construída sobre as estruturas de *selenium* e *Appium.*

Esta ferramenta engloba diferentes funcionalidades e para além do *Katalon Studio*, também pode ser utilizado o *katalon Recorder* (é um IDE no *chrome* ou *Firefox*, onde é possível registar e gerir um teste automatizado convertendo assim as ações em diversas linguagens de programação orientadas à automação, nomeadamente, C#, Java) e o *Katalon Analytics* (é uma aplicação web que oferece aos utilizadores uma visão mais completa do estado do teste, com o auxilio de painéis e relatórios) (katalon; Purushothaman et al., 2018; M. A. Umar & Zhanfang, 2019).

[1] Linguagem de programação de alto nível

[2] Linguagem de programação multiplataforma suportada por diversos sistemas operacionais (Linux,Windows,Solares)

Segundo Purushothaman et al. (2018) e M. A. a. C. Umar, Zhanfang (2019), o *Katalon Studio* contém as seguintes Características:

Vantagens

- Não existem taxas de licenciamento nem de manutenção;
- Fácil integração de estruturas e recursos para criação/Execução de testes;
- Fácil programação no que toca a estrutura de *Selenium*;
- Utilização flexível para a reutilização de dados de testes;
- Integração com ferramentas de gestão JIRA, Slack e outras;
- Execução paralela de testes.

Desvantagens

- Pouca documentação para que sirva de apoio;
- Suporte limitados no que toca a recursos;
- Linguagens de script com poucas opções, uma vez que só é compatível com Java e Groovy;
- Demora algum tempo na compilação e execução de testes;
- O projeto tem de estar no *Katalon Studio* para que este seja executado.

2.5.2 Robot Framework

Robot Framework é uma ferramenta de automação de testes open source, baseada em keyword-driven (permite serviços, dados e scripts serem independentes uns dos outros) (Na & Huaichang, 2015; Pajunen, Takala, & Katara, 2011) que foi desenvolvida com o intuito de elaborar testes a nível de aceitação (Bisht, 2013; Framework, 2020).

Esta ferramenta tem acesso a diversas bibliotecas, que vão contribuir para a integração de vários tipos de modelos. Para além dos modelos de integração, podem ser desenvolvidos scripts em Python e Java (Harsha & Kumari; Pajunen et al., 2011).

O *Robot framework* contém sintaxe fácil, e compreensível por qualquer pessoa, uma vez que, as *keywords* contêm uma ação descrita em linguagem percebida por todos. Para além disso, possui um rico ecossistema, uma vez que constitui bibliotecas e ferramentas que podem estar inseridas em projetos separados (Framework, 2020; Harsha & Kumari).

Na visão de Na and Huaichang (2015) e Pajunen et al. (2011), *Robot framework* contém as seguintes Características:

Vantagens

- Ferramenta Open Source;
- Framework genérico, podendo automatizar soluções web e APi;
- Fácil compreensão, e mais intuitivo devido á utilização de keywords;
- Documentação de fácil compreensão, tanto da robot framework, como das suas bibliotecas que poderão ser integradas nos testes;
- Logs e Reports são gerados de detalhada e organizada;
- Possível desenvolver scripts em Python e Java.

Desvantagens

- Necessita de um IDE para criação dos testes;
- Instalação exige que todos os pacotes, drivers e dependências das bibliotecas sejam instalados separadamente.

2.5.3 Protractor

Protractor, é uma ferramenta de testes end-to-end (testar fluxos/processos do inicio ao fim), segundo a informação presente no web site oficial de (Protractor) está focado na automação de testes em aplicações em Angular e Angular JS (Bandwal et al., 2019). Esta ferramenta, executa testes tendo interação com a página web através de um web driver, simulando assim, o comportamento do utilizador (angularjs; Bandwal et al., 2019; Fat, Vujovic, Papp, & Novak, 2016; Protractor).

Segundo a documentação oficial de angularjs e na visão de Bandwal et al. (2019), esta ferramenta não está direcionada somente na automação de testes em Angular, mas também visa a criação de testes de regressão no que toca a aplicações web desenvolvidas com outro tipo de linguagens de programação, podendo assim combinar tecnologias tais como *Selenium*, *Jasmine Web Driver*.

No ponto de vista de Bandwal et al. (2019) e Fat et al. (2016), a ferramenta *Protractor* contém as seguintes características:

Vantagens

- Usa uma sintax simples para desenvolvimentos de testes;
- Na codificação dos testes não é necessário adicionar paragens entre ações, pois, esta ferramenta só executa a próxima ação quando a página web termina as tarefas que estejam pendentes;
- Compatibilidade com diferentes browsers;
- Oferece suporte a estruturas como o Jasmine[3], mocha[4], cucumber[5];
- Execução de script em diversas máquinas é realizada de forma fácil.

Desvantagens

- Depuração de resultado dos testes bastante complicada uma vez que para ter mais controlo do teste, na codificação do teste terão de incluir capturas de ecrã;
- Suporta apenas Javascript, o que limita caso se pretenda fazer testes a diversos tipos de aplicações;
- Só é utilizado para aplicações web;
- Pouco suporte nos browsers, com excepção do chrome;
- Indicada apenas para testes end-to-end.

2.5.4 Watir

A ferramenta Watir é open source, sendo utilizada para a realização de automação de páginas web usando a linguagens de script Ruby (Gogna, 2014; Islam, 2016). Os testes desenvolvidos no Watir podem ser executador em diversos browsers, tais como, Internet Explorer, Firefox, Chrome e Opera, tendo foco testes funcionais, de sistema e de regressão (Islam, 2016; Singh & Tarika, 2014).

Ressalva-se que Watir não é uma ferramenta para gravação de casos de testes, uma vez que não contém a sua própria interface (Gogna & Kumari, 2011), usando um IDE externo

[3] Framework que dá suporte a estrutura de testes javascript
[4] Estrutura de teste JavaScript para programas Node.js
[5] Ferramenta de software que suporta o desenvolvimento orientado pelo comportamento

para escrever e executar os casos de teste, por exemplo o IDE RubyMine. Contudo, segundo Islam (2016) e Singh and Tarika (2014), esta ferramenta contém as seguintes características:

Vantagens

- Ferramenta Open source;
- Utiliza Ruby, que é uma linguagem que contém muitos recursos, sendo possível conexões a Base de dados, leitura e escrita de dados em ficheiros, exportação de xml e bibliotecas;
- Suporta vários browsers, mesmo os menos utlizados, o que facilita em aplicações web mais antigas;
- Utilização fácil e flexível.

Desvantagens

- É limitado a aplicações web;
- Pouca documentação de suporte.

2.5.5 Comparação da melhor ferramenta neste caso de estudo

As Figura 5 e Figura 6 apresentam em resumo, as vantagens e desvantagens existentes, em cada ferramenta escolhida para análise neste caso de estudo. As mesmas foram selecionadas porque todas continham uma vertente de testes web, algo que era essencial neste projeto.

Após análise teórica das quatro ferramentas, existiu um agrupamento em pares das ferramentas, por forma a agrupar as que continham mais semelhanças.

O Protractor e Watir (Figura 5) são ambos *open source*, suportam vários tipos de browsers, sendo de fácil utilização. Em contrapartida, estão limitados a aplicações web, o que revelaria um problema neste projeto, uma vez que os casos de testes englobam outro tipo de aplicações. Estas duas ferramentas, Protractor e Watir, não têm muita documentação e suporte, por este motivo foram excluídas para a parte prática do projeto.

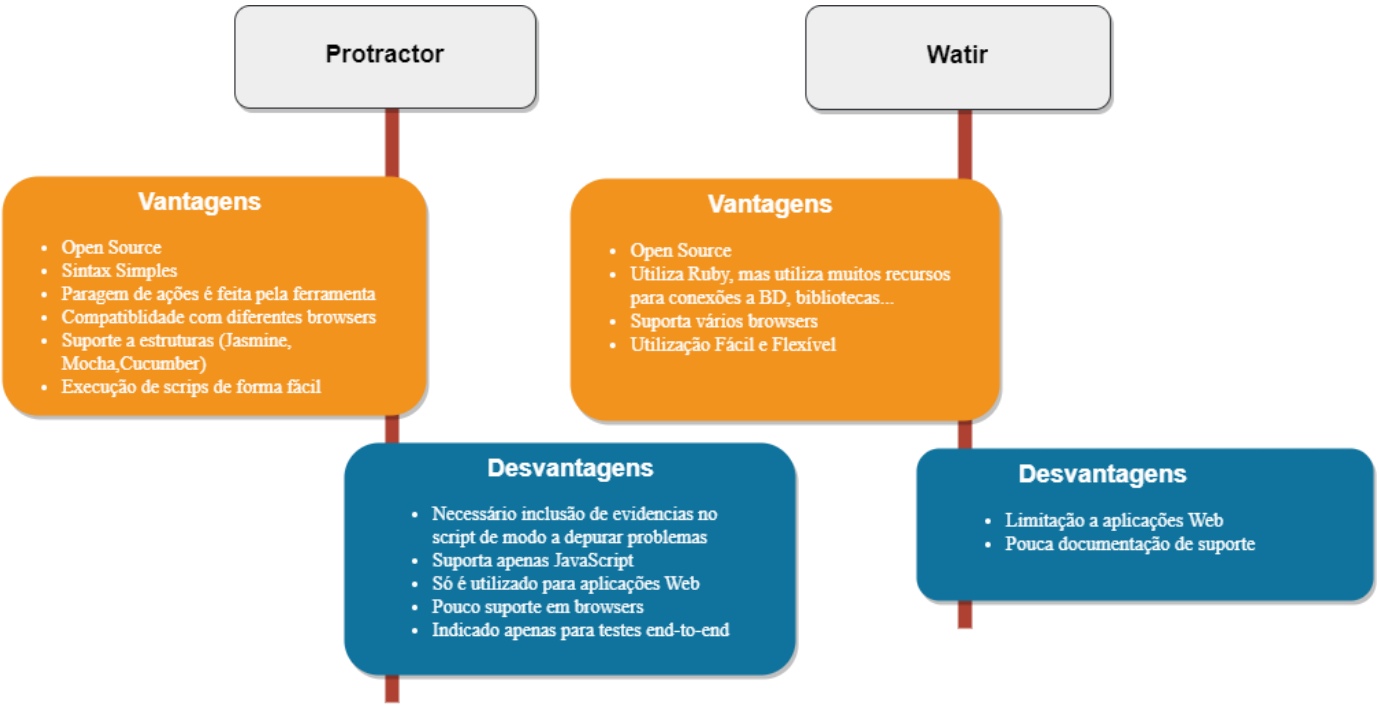


Figura 5 - Vantagens e Desvantagens Protactor e Watir

Quanto ao *Katalon Studio* e *Robot framework* (Figura 6), ambas a ferramenta contêm uma estrutura de *Selenium*[6], geram relatórios html em cada execução, são de fácil compreensão. *Katalon Studio* ganha a *robot framework* na instalação, pois a instalação do *Katalon* contém todas as dependências necessárias, enquanto que na *robot framework* é necessário instalar todos os pacotes, drivers e dependências bibliotecas de forma isolada.

Há um ponto bastante importante, e foi por essa mesma razão que a escolha foi direcionada para a *robot framework*, o desempenho. *Katalon Studio* é uma ferramenta baseada em *Groovy*, leva assim mais tempo na compilação e execução de testes, enquanto que a *robot framework* é baseado em *python*, que é uma linguagem mais leve e, consequentemente, leva a um melhor desempenho na compilação e execução de testes.

Como o objetivo, recai na criação e execução diária de baterias de testes, evoluindo assim os casos de teste ao longo do tempo, é importante pensar no ritmo de execução dos mesmos, quanto mais rápido compilarem e executarem, mais rápido começarão outros casos de teste, sendo assim possível **testar mais** em **menos tempo**.

[6] Estrutura portátil para testar aplicações da web

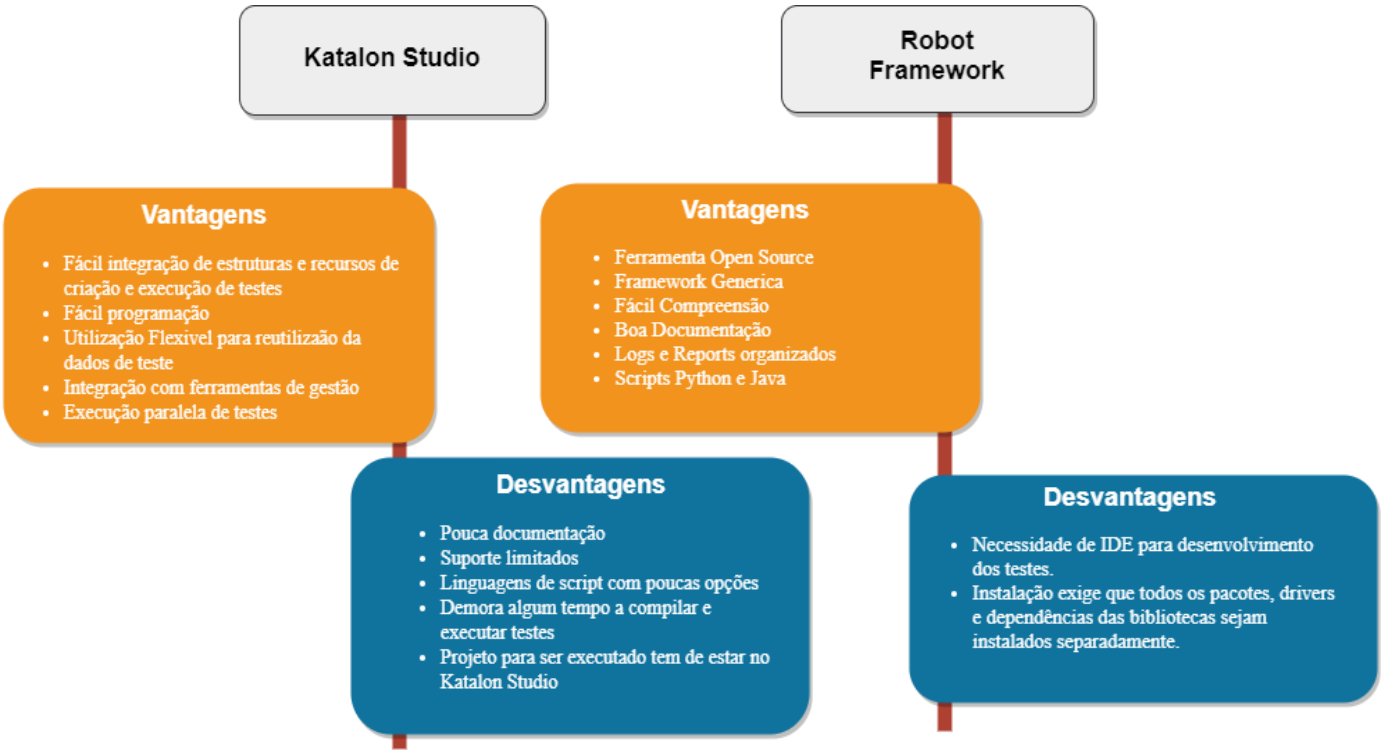


Figura 6 - Vantagens e Desvantagens Katalon Studio e Robot Framework

3. Caso de Estudo

Neste capítulo é abordado o caso de estudo realizado na empresa SIBS, pretendendo-se apresentar pormenorizadamente o problema e a solução implementada.

Os casos de teses serão devidamente especificados, de modo a obtermos uma visão dos requisitos pretendidos em cada um deles. Em termos de codificação, será abordado o tema das boas práticas, assim como a definição de *Keywords*. Para além dos temas indicados, será ainda descrita a execução de testes, incidentes na criação de testes e baterias de execuções diárias.

3.1 Enquadramento

Como referido no capítulo 1.1, existem atualmente mais de 300 milhões de utilizadores dos serviços financeiros disponibilizados pela empresa em estudo SIBS, desta forma e tendo em consideração o constante desenvolvimento de novas funcionalidades, modificações, melhorias e correções é de extrema importância avaliar e testar o SW, minimizando assim os possíveis impactos destas intervenções.

Considerando todos os serviços e funcionalidades disponibilizadas a todos os utilizadores, torna-se humanamente impossível garantir uma boa qualidade e quantidade de testes a serem implementados, assim o recurso a ferramentas de testes torna-se, diria, obrigatório de forma a assegurar a entrega para produção. Deste modo, este projeto tem como intuito avaliar o modo

de como as ferramentas de testes de software permitem minimizar os erros encontrados no processo de desenvolvimento do mesmo, inerente ao setor do IPS (*Instant Payments Solution*). Para além de avaliar ferramentas, foi desenvolvido um projeto de automação de testes, existindo foco em diversas aplicações desenvolvidas pela equipa do IPS.

Com o desenvolvimento destes casos de testes automatizados, é possível executar os testes diariamente, com o objetivo de validar se existem erros nos processos designados por estáveis. Tais erros podem surgir com os novos desenvolvimentos e correções de funcionalidades. Com estas execuções diárias, conseguimos ter perceção da existência de possíveis erros numa maior escala de abrangência, face à realização de testes manuais, o que era impensável a realização de testes manuais a todos os processos estáveis diariamente.

Com o aumento de execução de testes, será dada uma maior segurança em termos de entrega de software, uma vez que os erros serão identificados mais cedo, possibilitando a execução de correções necessárias antes do software ser enviado para outras equipas de validação.

3.1.1 Metodologia e planeamento do projeto

A organização do projeto é essencial, tendo sido para isso aplicada uma metodologia ágil denominada de *srum*, uma vez que é a utilizada atualmente pela equipa, sendo igualmente um requisito da organização onde decorre o caso de estudo.

As metodologias ágeis estão focadas no trabalho em equipa, sendo este um trabalho criativo com o objetivo de existir uma maior eficácia e acessibilidade. (Highsmith & Cockburn, 2001) Este trabalho está centrado no desenvolvimento interativo, adaptativo e evolutivo, promovendo assim, uma resposta rápida e flexível à mudança (Larman, 2004).

Com o objetivo de aplicar esta abordagem, serão respeitados determinados valores:

"- Indivíduos e interações sobre processos e ferramentas

- Software que trabalha sobre uma documentação completa

- Colaboração do cliente sobre negociação de contrato

- Respondendo a mudanças após seguir um plano" (Abrahamsson, Salo, Ronkainen, & Warsta, 2017)

O *scrum* é um modelo empírico, sendo que quem usa este método adquiriu conhecimentos de uma experiência vivida, tomando decisões com base em conhecimento (Layton, 2015). Com este método, as equipas terão uma comunicação regular, podendo falar e ser ouvidas com realismo, tendo como objetivo, o compromisso com metas compartilhadas (Audy, 2015). No *Scrum* é criado um *backlog* de produto, onde estão presentes os recursos/requisitos necessários para o projeto. Estes requisitos estão inseridos com valor de prioridade, de modo a trabalharmos com as prioridades mais altas primeiro e assim quando acabam os recursos, as tarefas não concluídas estarão com prioridades mais baixas (Rubin, 2012).

Existem três pilares praticados no *scrum*:

- **Transparência**: Todos os elementos devem estar focados no projeto de igual forma.

- **Inspeção**: Após existência de uma oportunidade ou risco, todos colaboram de modo a garantir uma melhor solução.

- **Adaptação**: Todos tem capacidade de adaptação, participando nos planos de ação (Audy, 2015).

Em termos práticos serão planeados *sprints* (ciclo de desenvolvimento do *scrum*). Cada um terá duração de duas semanas. No planeamento do Sprint são definidas todas as tarefas que já se encontrem estáveis, ou seja, tarefas em que os requisitos estão completamente definidos e esclarecidos. Com este *backlog*, será possível desenvolver os testes à medida que o software é desenvolvido. Isto levará a uma maior organização e partilha de requisitos entre o programador e o tester. As tarefas em *backlog* passam por quatro diferentes estados: "**To do**", "**doing**", "**testing**", "**done**" (Sims & Johnson, 2012).

Para além das reuniões de preparação do sprint, todos os dias haverá uma reunião de *scrum* em que se explica o que cada elemento da equipa esta a fazer, assim como, dificuldades e bloqueios existentes. No final de cada sprint, existe uma reunião com o objetivo de verificar os avanços existentes no projeto. Estas reuniões foram realizadas de quinze e quinze dias, com a equipa de desenvolvimento do IPS.

3.1.2 Arquitetura geral da solução

Na Figura 7 é possível verificar a arquitetura desenvolvida para solucionar o problema existente neste caso de estudo. Esta arquitetura define a forma como as aplicações estão ligadas à *framework* de automação de testes.

No lado esquerdo são definidas todas as aplicações que estão a ser testadas até ao momento no ISP (FTM, Sistema, Portal IPS, Cognos e base de dados). Na componente do FTM está presente a MQ sendo o servidor de mensagens e o JMS responsável para realizar o transporte das mesmas.

O *Pycharm* é o IDE escolhido e contém as diversas ferramentas necessárias para o desenvolvimento dos casos de teste, sendo elas:

- robot framework
- python
- selenium
- powerShell

O *Jenkins* enquadra-se nesta solução, pois permite testar todos os componentes do IPS e possibilita uma execução automática, de um conjunto de operações essenciais, de modo a garantir o bom funcionamento do SW.

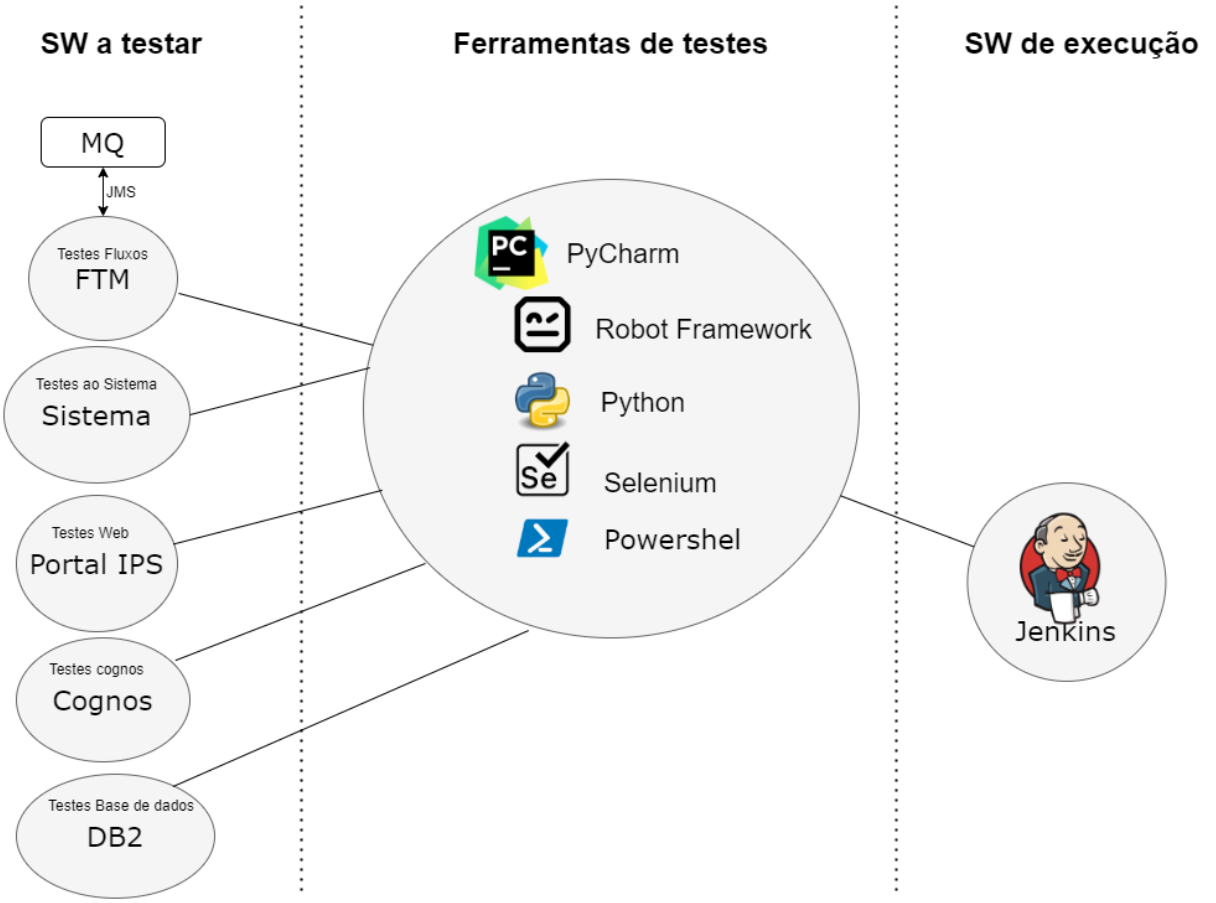


Figura 7 - Arquitetura da solução

3.2 Desenvolvimento do projeto

Neste capítulo são abordados diversos temas que estão ligados com a parte prática do projeto, sendo igualmente indicadas as ferramentas necessárias para o desenvolvimento de testes, bem como uma breve explicação e configuração do *jenkins*.

É apresentada uma explicação prática dos componentes do IPS e os testes que integram em cada um deles. A estrutura da robot framework também é mencionada, assim como a execução e configuração de baterias de testes. No que diz respeito à análise e levantamento de requisitos, foi necessária uma reunião para debater o tema.

3.2.1 Instalação de ferramentas de desenvolvimento

Nesta secção serão mencionadas as ferramentas e librarias necessárias para construir um ambiente de desenvolvimento de testes.

Inicialmente deve ser instalado o python[7]-2.7.15.amd64 (sendo uma linguagem de script compatível com a *robot framework*) na seguinte diretoria (C:\Python27). Com o decorrer da instalação deve selecionar a opção da criação das variáveis de ambiente. Com a adição destas variáveis de ambiente, quando um *script* de extensão .py é executado via linha de comandos, consegue saber diretamente o caminho para o interpretador Python de modo a conseguir decifrar o código.

Seguidamente é necessária a instalação de um IDE (Ferramenta para escrever código) uma vez que a *robot framework* não possui uma interface. De modo a combater a falta de interface, será instalado o *Pycharm*. Este IDE contém bastantes funcionalidades que ajudam a desenvolver e executar o código.

Após instalação do IDE, será necessário proceder á instalação de librarias.
Esta contém um conjunto de *keywords*, que tem como objetivo executar determinadas funções. As librarias têm de ser instaladas (**python setup.py install**) na seguinte ordem:

- robotframework-3.0.4
- selenium-3.8.1
- decorator-4.4.0
- robotframework-selenium2library-1.8.0
- robotframework-extendedselenium2library-0.9.1

[7] Linguagem de programação de alto nível

- robotframework-databaselibrary (1.2.4)
- ibm_db-3.0.1

Após realizadas as instalações das librarias mencionadas anteriormente, para efetuar a integração da *robot framework* no *PyCharm* é necessário instalar o plugin intellibot.

Em termos de execução de teste, é necessário adicionar e configurar *external tools (*Figura 8 e Figura 9*)*. Estas *external tools* contém informação para execução de testes, no qual será indicado o .bat da robot para execução, quais argumentos serão passados, com indicação da diretoria.

Neste caso, são configuradas as seguintes *external tools* para execução local, de modo a ser utilizada no desenvolvimento e depuração de problemas.

Ambiente de DEV:

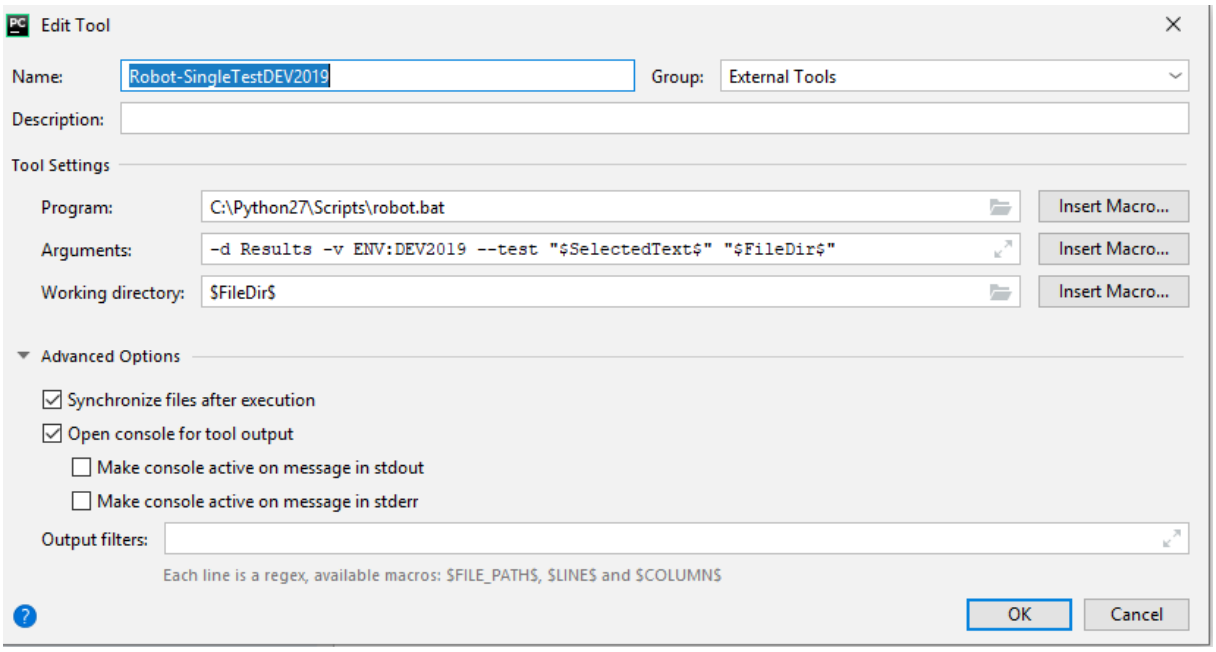


Figura 8 – External Tools DEV

Ambiente TST:

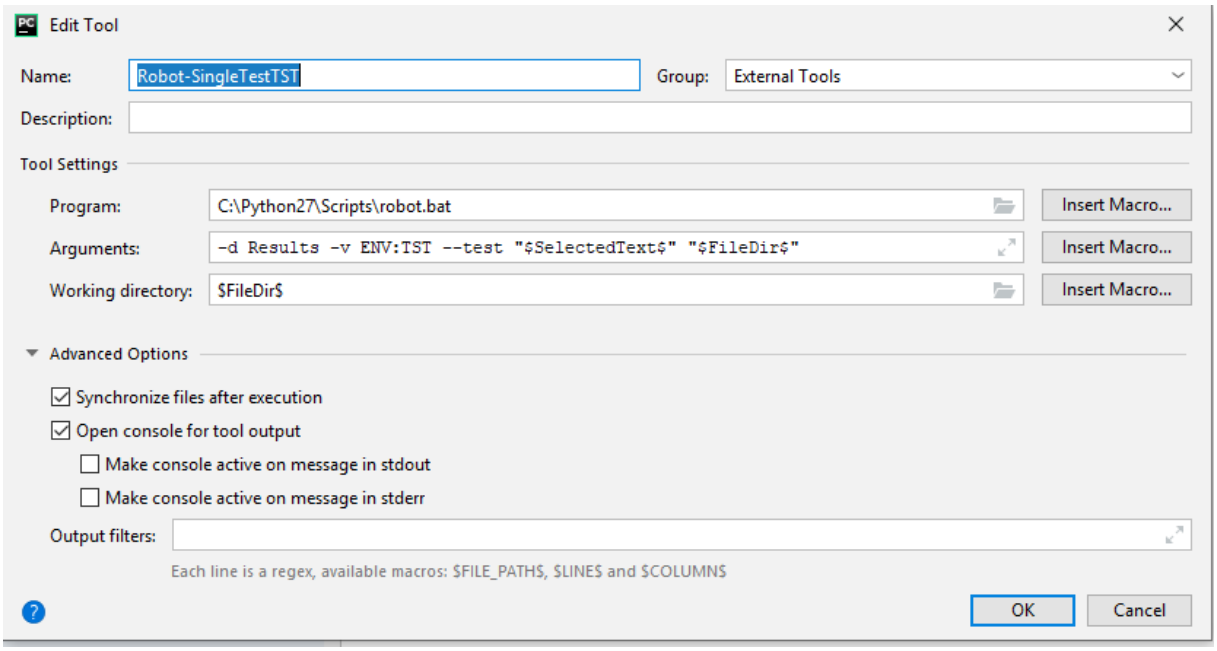


Figura 9 - External Tools TST

Uma vez que serão desenvolvidos casos de teste a páginas web, é necessário realizar essa configuração. Será adicionado um *webDriver* à pasta onde está instalado o browser.

O *webDriver* permite execução direta de testes via código, emparelhando-se com a API do *selenium* e permite aceder ao documento *DOM* completo da página *HTML*, para manipulação de propriedades e fumões e com isto, irá realizar as ações introduzidas no código.

Dentro do pycharm terá de definir a pasta onde colocou o webBrowser (Tools->Web Browsers). Será necessário ligação à base de dados, para isso, é necessário adicionar o odbc driver,assim como, adicionar a variável de ambiente IBM_DB_HOME. Para além dos requisitos mencionados anteriormente, devido ao controlo de versões, é necessário adicionar o projeto ao repositório, neste caso, será o GIT.

3.2.2 Servidor de integração contínua Jenkins

A integração continua, em conjunto com ferramentas de automação traz bastantes benefícios pois o trabalho de cada indivíduo da equipa é integrado com mais frequência, o que leva, a uma redução de erros. Muitas vezes, são feitas diversas questões, nomeadamente:

Quanto tempo demora a correção de um requisito (desenvolvimento e testes) e este entrar em produção? Poderíamos reduzir os problemas se fossem detetados antes? A última alteração daquele elemento da equipa já está validada? A integração continua, ajuda a simplificar muitos dos problemas descritos (Smart, 2011).

Os erros são detetados de forma rápida e existe uma redução de riscos, uma vez que num sistema deste tipo a integração é rápida e continua (Fowler & Foemmel, 2006). Com estes benefícios a entrega de software irá ser mais fácil, rápida e com menos bugs (Smart, 2011).

Existem diversas ferramentas no mercado para realização de integração contínua e neste projeto foi utilizado o *Jenkins*, uma vez que é um software já utilizado na equipa do IPS.

Jenkins é uma ferramenta de software livre escrita em java que suporta a construção e testes de software (Seth & Khare, 2015; Smart, 2011). É de fácil utilização, tendo uma interface simples e intuitiva, logo, o utilizador não vai perder muito tempo a perceber como a ferramenta funciona. A partir de download de plug-ins esta ferramenta cobre as necessidades do utilizador, podendo usar para sistemas de controlo de versões, notificações de email, integração com sistemas externos.

Para além de estar integrado na equipa do IPS, o *jenkins*, foi selecionado para o projeto devido aos benefícios descritos anteriormente.

Para instalação do *jenkins*, é necessário seguir os passos descritos no anexo 2.

3.2.3 Estrutura do código

Na robot framework, existe uma estrutura de código dentro de cada script de teste (Figura 10).

Dentro do script principal temos os ***Settings***, onde é realizada as chamadas de ficheiros .py, importação de bibliotecas, e ficheiros auxiliares.

O utilizador poderá codificar estas chamadas no início do script, e não será necessário chamar por exemplo, uma biblioteca em cada página que esta seja usada. As variáveis de input ou variáveis auxiliares do teste são configuradas na secção **Variables**. A secção dos ***Test cases*** é a mais importante, uma vez que definimos o nome do teste, *keywords* a chamar, e sem esta secção o teste simplesmente não executa.

As ***keywords***, são palavras chave, que podem representar outras *keywords*, que realizam ações sobre o teste.

A secção de ***Documentation*** deve ser sempre preenchida, dando suporte, na explicação do teste case e *keywords*. Com este tipo de suporte, conseguimos perceber de forma autónoma como funciona o teste, e os passos que são realizados na execução do mesmo, assim, será mais fácil aplicar alterações e melhorias.

```
*** Settings ***
Library     Selenium2Library
Library     BuiltIn
Resource    ../../IPS/FTM/KwMensagens.robot


*** Variables ***
# Script a chamar para execução do teste via powerShell
${pathTest}             C:\\\\IBM_Work\\\\'Automacao de testes'\\\\TestMesssages\\\\TestChain\\\\01_Test\\\\test_chain.ps1
${BANK1}        AUTVQLXGFQD     # Bic Banco ordenante
${BANK2}        AUTJQPVCHVY     # Bic Banco Beneficiario
${AMOUNT}       1.01            # Montante a ser enviado para o banco beneficiario
${TestNumber}   001             # Número do teste a executar


*** Test Cases ***
Criacao e validacao de uma CT entre Direto e outro Direto
    [Documentation]     Envio de uma CT do banco Direto para outro banco direto
    ...                 Transação que chegou por parte do banco beneficiário fica no estado concluído com sucesso.
    ...                 A transação de aceitação por parte do banco beneficário foi entregue ao banco ordenante com sucesso.
    ...                 Banco benficiário recebe o ack enviado pela Sibs.
    ...                 Existe uma reserva antes do envio da nota de transferência para o banco beneficário
    ...                 Existe movimentação de saldos de uma conta para  a outra no final do processo.

    Criacao e validacao de uma CT       ${BANK1}        ${BANK2}        ${pathTest}     ${ENV}      ${TestNumber}       ${AMOUNT}
```

Figura 10 – RF Estrutura do código

Para além da estrutura do código, neste este projeto foram adaptadas boas práticas de programação, especialmente:

- código bem documentado (facilita na leitura do teste e no processo de desenvolvimento)
- definição de declaração de variáveis (com regras de nomes de variáveis o código está mais coeso, e organizado)
- definição de regras para dar nomes a *keywords* (Para melhor interpretação do que a *keyword* faz terá de conter pelo menos 3 palavras e que uma ou várias represente uma ação).

Estas boas práticas, foram definidas numa reunião de *scrum* pelas pessoas que tinham contributo no projeto de automação de testes, sendo adicionadas posteriormente ao projeto, para que fosse possível, existir um código limpo, eficiente e de simples compreensão.

Uma das regras é a utilização de nomes significativos para as *Keywords* e para os nomes dos testes, com isto um utilizador que não conheça nem o código nem o processo do teste, consegue perceber os passos de maneira mais simples. Para além disto, cada *keyword* tem a secção [*Documentation*], onde é esclarecido o que vai acontecer naquela Kw ou no caso de teste.

3.2.4 Explicação prática dos tipos de testes

Neste subcapítulo serão abordados os vários tipos de testes desenvolvidos no projeto IPS. Será compreendido o objetivo de desenvolver cada tipo de teste, as tecnologias usadas em cada componente (FTM, *Message Queue*, *Cognos*, DB2, portal, sistema) existindo uma explicação detalhada do código chave de cada género de teste.

No IPS, existem diversos modelos de fluxos transacionais, que podem ou não realizar movimento financeiro e originar alterações de saldos. Em qualquer um dos fluxos, o IPS é sempre intermediário, não havendo comunicação direta entre bancos. Para enquadramento vai ser descrita uma breve definição desses fluxos.

Um **pedido de transferência bancária** trata-se de uma transferência bancária de um banco ordenante para um banco beneficiário. O fluxo envolve uma mensagem de pedido de transferência a partir do banco ordenante, ficando o valor a transferir cativo e retirado do saldo disponível até que o banco beneficiário, responda com a aceitação ou rejeição do pedido de transferência. Em caso de resposta positiva, o saldo cativo do banco ordenante é retirado e é creditado no saldo disponível do banco beneficiário; em caso de rejeição, o saldo cativo volta ao saldo disponível do ordenante.

Sobre um pedido de transferência, o banco ordenante pode realizar um **pedido de investigação**. Esta é uma operação que não origina movimentos financeiro e serve para o banco ordenante obter informação do estado do pedido de transferência. Em caso de algum erro técnico no envio da mensagem ao banco ou que a transação não tenha ficado concluída, é enviada ao banco ordenante uma mensagem com a informação de que o IPS dispõe, seja ela a resposta positiva ou negativa ao pedido de transferência.

É permitido ao banco ordenante, realizar um **pedido de devolução de fundos** (***Recall***) sobre um pedido de transferência, até treze meses após a realização da mesma e é realizada pelo banco. Esta operação não realiza movimento financeiro e está sujeita a aceitação/rejeição

por parte do banco beneficiário, podendo o mesmo nem responder à Recall. Só em caso de resposta positiva por parte do banco beneficiário é que haverá movimento financeiro.

As **respostas a *Recall*** são fluxos de resposta aos pedidos mencionados anteriormente e são enviadas pelo banco beneficiário. A resposta pode ser positiva e originar movimento financeiro para o banco ordenante do valor do pedido de transferência original subtraído de uma taxa a definir pelo banco beneficiário; ou ser negativa e daí não originar alteração de saldo.

Um **pedido de devolução de fundos solicitada pelo banco ordenante (*Recall by the originator*)**, pode ser iniciado dentro de treze meses após a transferência bancária e é despoletada não pelo banco como uma *Recall*, mas sim, pelo cliente do banco que originou a transferência original, devido a motivos diferentes, valor errado, IBAN incorreto. Mesmo sendo um pedido de devolução feito pelo ordenante, a decisão de devolução é sempre tomada pelo banco beneficiário.

O **pedido de status** é semelhante ao pedido de investigação, sendo que o pedido de status é feito sobre um pedido de devolução ou pedido de devolução pelo Ordenante. Se o IPS possuir a resposta positiva ou negativa, esta é devolvida como resposta ao pedido de status. Se não, informa que o pedido de devolução se encontra pendente de resposta.

3.2.4.1 Especificação de casos de testes

Antes de iniciar o desenvolvimento de casos de testes, é necessário proceder á sua especificação. A fase de especificação de testes, é necessária pois, é nesta fase que são definidas as tarefas mais importantes, ou as tarefas que englobam mais processos de modo a criar as condições necessárias para ser possível testar todos os cenários possíveis (Stocks & Carrington, 1993).

Com uma boa especificação, garantimos a execução de testes, que simulam de forma mais fiável as ações humanas sobre um determinado software, com o objetivo de garantir que que este se encontra com o comportamento normal.

Para além disso, com uma boa bateria de testes, conseguimos antecipar problemas no código especificado, verificando se os novos desenvolvimentos afetam de alguma forma, funcionalidades existentes evitando assim, problemas de regressão. Além disso, é possível agendar testes calendarizados ou de carga, para testar a capacidade física das máquinas que suportam toda a infraestrutura.

Para a realização do documento de especificação de testes foi efetuada uma reunião antes do início do projeto, com dois elementos sénior da equipa, de modo a validarmos quais os pontos que precisávamos para especificarmos casos de testes no IPS. Nesta reunião, a partir das necessidades do IPS, inicialmente foram definidas as aplicações que careciam de casos de teste:

- Portal e *cognos*, uma vez que é uma plataforma utilizada diretamente pelos bancos, sendo importante que estes realizem as suas tarefas sem a existência de erros.
- FTM, onde contém os fluxos de todo o tipo de transações realizadas no IPS, sendo relevante testar, validando que cada fluxo segue as etapas definidas.
- Testes de validação do sistema, para serem garantidas as condições necessárias para o software executar sem falhas e com tempo de respostas rápidas.
- Testes à base de dados, com a importância de validar os dados inseridos na base de dados, face aos dados enviados nos pedidos de transações, sendo essencial que estejam coerentes.

Após definição das aplicações, foram discutidos os casos de teste que deveriam ser implementados. A escolha destes foi realizada pelos elementos sénior que se encontravam na reunião, devido a terem um maior conhecimento dos processos chave e repetitivos existentes na equipa do IPS. Após esta escolha, iniciou-se a especificação dos casos que iriam ser desenvolvidos, este trabalho foi realizado pelo mestrando com apoio dos elementos sénior, com o objetivo de fazer conhecer melhor o software e processos que existem no IPS.

Uma vez que, há necessidade de desenvolver testes em diversas aplicações, foi adicionado o campo "**Tipo**" à especificação, que informa o tipo de teste (Fluxo, Sistema, Portal, *Cognos*, Base de dados).

Outro campo necessário é o "**Cenário de teste**", no qual é descrito de forma sucinta o que faz o teste, dando um nome ao mesmo. Já no campo "**Descrição do cenário**", é adicionado de forma detalhada todo o processo necessário para a realização do teste.

A execução dos testes pressupõe algumas pré-condições, que são mencionados na coluna "**Condições necessárias**" no documento de especificação, que indica os processos/ações que devem ser realizadas antes da execução do teste. A título de exemplo, um teste que realize uma transferência entre o banco A e o banco B, necessita de pré-condições, nomeadamente:

- O banco A e o banco B têm de estar registados no sistema,

- O banco A necessita de ter saldo suficiente para a realização da transferência.

Sendo importante a realização da análise do output do teste, foi criado o campo "**Resultado Esperado**". Este fornece o poder de decisão sobre o sucesso ou não do caso de teste. Pode possuir lógica mais ou menos complexa, dependendo da exigência da especificação. Num pedido de transferência, o sucesso da operação pode ser definido apenas pelo sucesso no envio das mensagens no fluxo, mas a análise deve verificar não só o sucesso no envio das mensagens, como deve validar os saldos disponíveis inicial e final de cada banco e o saldo cativo. Não só a especificação do teste é importante como também a correta análise dos resultados de modo a garantir confiança no produto.

No Quadro 1, é possível ver um exemplo da especificação de testes realizada no projeto, no Quadro 2 do anexo 3, é possível analisar todos os casos de teste.

Tipo	Cenário de teste	Condições Necessárias	Descrição do Cenário	Resultado Esperado
Fluxo - Transferências	Transferências com sucesso entre Direto e outro banco Direto	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Participantes com saldos disponíveis para realizar a operação - Bancos sem indisponibilidade agendada	O Banco ordenante deverá enviar um pedido de transferência, através do envio de uma mensagem de nota de transferência. Quando a mensagem chegar ao banco beneficiário, este deverá responder com uma resposta de ack positiva. Esta mensagem deverá ser processada e integrada no sistema para que depois possa ser entregue ao Banco ordenante, no final deverá ser enviado um ack de volta para o BB.	Transação que chegou por parte do banco beneficiário fica no estado concluído com sucesso. A transação de aceitação por parte do banco beneficiário foi entregue ao banco ordenante com sucesso. Banco beneficiário recebe o ack enviado pela Sibs. Existe uma reserva antes do envio da nota de transferência para o banco beneficiário Existe movimentação de saldos de uma conta para a outra no final do processo.

				Verificação do envio da mensagem para a BAM
Fluxo - Transferências	Transferências com sucesso entre Directo e o seu Indirecto	- Bancos registados na aplicação IPS - Banco Indirecto com ligação ao banco directo - Ambos participantes têm de estar num estado que permita realizar a operação - Participantes com saldos disponiveis para realizar a operação - Bancos sem indisponibilidade agendada - Indirecto com CREL definido	O Banco ordenante deverá enviar um pedido de transferência, através do envio de uma mensagem de nota de transferência. Quando a mensagem chegar ao banco beneficiário, este deverá responder com uma resposta de ack positiva. Esta mensagem deverá ser processada e integrada no sistema para que depois possa ser entregue ao Banco ordenante, no final deverá ser enviado um ack de volta para o BB.	Transação que chegou por parte do banco beneficiário fica no estado concluído com sucesso. A transação de aceitação por parte do banco beneficiário foi entregue ao banco ordenante com sucesso. Banco beneficiário recebe o ack enviado pela Sibs. Não existe uma reserva antes do envio da nota de transferência para o banco beneficiário Existe movimentação na conta técnica do indirecto e na conta do directo, contudo o salto total agregado do Directo não pode alterar (BALANCE_V) Verificação do envio da mensagem para a BAM

Quadro 1 – Exemplo de especificação dos casos de teste

3.2.4.2 Testes de fluxos

Os fluxos de movimentações de saldo indicados no ponto 3.2.4 estão inseridos na ferramenta FTM. O FTM é uma plataforma composta por vários componentes e fornece, uma estrutura para realizar integração de aplicações e serviços direcionados para o mercado financeiro ("Financial Transaction Manager,") .

Nesta plataforma é possível ver o fluxo do processos de transferências bancárias (Figura 11), tendo em conta todas as mensagens (.xml) enviadas e recebidas, de forma a ser possível validar se o processo fica concluído ou com erros. As mensagens são enviadas por recursos JMS definidos, o FTM suporta o uso do WebSphere MQ[8] como transporte JMS [9].

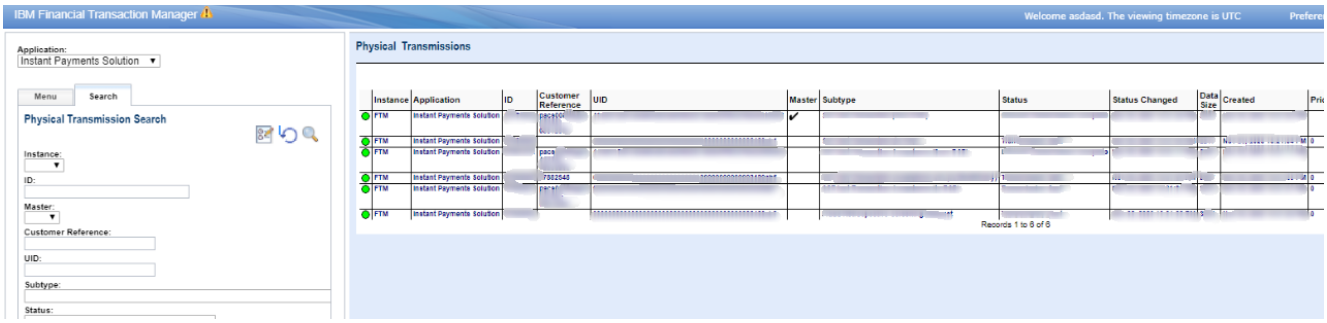


Figura 11 – Exemplo de um fluxo de mensagens no FTM

É essencial a realização de testes aos fluxos de processos estáveis, de modo a ser possível perceber se por exemplo o fluxo ligado a uma transação bancária se encontra com todas as etapas que deveria. No IPS já existia uma plataforma desenvolvida em *powershell* [10], onde é possível escolher o fluxo (Pedido de Transferência,Recall, etc.) e o mesmo cria as mensagens necessárias a partir de mensagens tipo (Templates), sendo possível executar o teste com variáveis de entrada, tendo valores configuradas ou novos.

Além disso também permite, definir o endereço do servidor de mensagens de modo a saber qual é o servidor selecionado para injeção de mensagens. De forma a reutilizar a ferramenta, e para evitar que novos casos de teste ou correções tivessem que ser mantidas em

[8] Sistema de mensagens para aplicativos
[9] Java Message Service – Permite que aplicativos clientes JMS comuniquem com serviços de mensagens do provedor
[10] Shell desenvolvida pela Microsoft, (. NET framework) incluindo uma linha de comandos e uma linguagem de script.

duas plataformas diferentes, foi adicionado um script em *robot framework* usando *python*, como se pode ver na Figura 12 que integra a chamada em *powershell* dos testes.

Desta forma o mesmo script é partilhado, sendo possível executar todos os casos de teste configurados.

Foi necessário efetuar algumas alterações ao script em *powershell* que lhe concedeu mais flexibilidade, como por exemplo novas variáveis de entrada. Para identificar na base de dados e FTM as mensagens enviadas pela *robot framework*, foi adicionado o texto "AUT" num campo identificador das mensagens.

```
def Call_PowerShell_Tests(Path,Environment,TestID, BANK1, BANK2, AMOUNT):
    # Teste para chamar script em powerShell de modo a executar os casos de teste existentes
    idTest = "'" + TestID + "'"

    if AMOUNT == "":

        logger.console("****** instrucao if-->")
        logger.console("" + Path + " -Environment " + Environment + " -Bank1 " + BANK1 + " -Bank2 " + BANK2 + " -TestCases " + idTest + "")

        p = subprocess.Popen(["powershell.exe",
                              " " + Path + " -Environment " + Environment + " -Bank1 " + BANK1 + " -Bank2 " + BANK2 + " -TestCases " + idTest + ""],
                             stdout=subprocess.PIPE)
    else:

        logger.console("****** instrucao else-->")
        logger.console("" + Path + " -Environment " + Environment + " -Bank1 " + BANK1 + " -Bank2 " + BANK2 + " -AMOUNT " + AMOUNT + " -TestCases " + idTest + "")

        p = subprocess.Popen(["powershell.exe",
                              " " + Path + " -Environment " + Environment + " -Bank1 " + BANK1 + " -Bank2 " + BANK2 + " -AMOUNT " + AMOUNT + " -TestCases " + idTest + ""],
                             stdout=subprocess.PIPE)
        logger.console("****** instrucao else 2-->")

    '''
    p = subprocess.Popen(["powershell.exe",
              " "+Path +" -Environment " + Environment + " -Bank1 " + BANK1 +" -Bank2 " + BANK2 + " -TestCases " + idTest + ""], stdout=subprocess.PIPE)
    '''

    out, err = p.communicate()
    list = out.split('MsgBox\Inbox:')

    return(list[1])
```

Figura 12 – Chamada em *python* de script em *PowerShell*

A Figura 12 representa um modelo de uma chamada, de um caso de testes que será despoletado via script *powerShell*. Nesta chamada, são enviados os parâmetros de entrada necessários para a execução do mesmo, e no fim da execução é retirado a partir do *python* o resultado dessa chamada, retirando assim os dados das mensagens enviadas, para adicionar os mesmos ao resultado do teste.

Uma vez que não é obrigatório o preenchimento, de todos os parâmetros de entrada do teste, foi adicionado uma validação na variável "*AMOUNT*", caso este esteja preenchido, é enviado como parâmetro na chamada do script, caso contrário, só são enviados os outros parâmetros de entrada.

3.2.4.3 Testes ao Sistema

Para além da necessidade de testar software, temos de ter em consideração as condições do local/sistema onde o software se encontra. Não queremos ter um software de confiança, se não temos um sistema equivalente, ou seja, é essencial que o sistema nos dê também essa confiança.

Para isso, é relevante o desenvolvimento de testes ao sistema, de modo a garantir, que este tem espaço livre para que não tenha falhas nem tempo de respostas muito longas durante a execução de processos. Para além destes pontos, o sistema também deve garantir que o seu comportamento é igual ao esperado, tendo em conta requisitos específicos de cada sistema.

Para garantirmos a qualidade do sistema, forma desenvolvidos testes de validação:

- Espaço livre no sistema;
- Validação da existência de *CoreDumps*;
- Remoção de *CoreDumps*.

Para desenvolver estes tipos de testes, foram adicionadas funções/Kw's chave utilizando o *Python*, como por exemplo, a realização do login no sistema via SSH (Figura 13), e a execução de diferentes comandos dentro do sistema, os comandos simples (Figura 15) e os complexos (Figura 14).

```
def Login_SSH(ssh_host, ssh_user, ssh_psw):
    #logger.console()
    proxy = None
    client = paramiko.SSHClient()
    client.load_system_host_keys()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(ssh_host,username=ssh_user,password=ssh_psw, sock=proxy)

    return client
```

Figura 13 - Login no Sistema via SSH

```python
def Executar_Comando_Complexo_SSH (ssh, cmd_command,passwordSSL):

    client_stdin, client_stdout, client_stderr = ssh.exec_command("su - mqm -c " + cmd_command, get_pty=True)

    time.sleep(0.2)
    client_stdin.write(passwordSSL + '\n')
    client_stdin.flush()

    flag= 0

    for line in client_stdout.read().splitlines():
        logger.console(line)
        if "running" in line:
            flag = 1

    return flag
```

Figura 14 - Código de execução comandos Complexos SSH

```python
def Executar_Comando_Simples_SSH(client, cmd_command):

    client_stdin, client_stdout, client_stderr = client.exec_command(cmd_command)
    data = client_stdout.read()
    output = data.decode()
    error = client_stderr.read().decode('utf-8')
    return output
```

Figura 15 - Código de execução comandos Simples SSH

Foi utilizado o modulo *Paramiko* (interface em *python* pura em tornos de conceitos de ssh), para a realização de login por SSH.

Nas máquinas usadas no IPS, apenas alguns utilizadores, estão autorizados a executar determinados scripts o que levou à criação de comandos complexos, que recebem o utilizador com o qual se pretende executar o comando enviado. Para outros comandos, que possam ser executados pelo utilizador de login, são usados os comandos simples.

3.2.4.4 Testes web

Dentro da equipa do IPS existe uma plataforma web direcionada para os bancos, e para a gestão interna da SIBS.

Se um banco entrar nesta plataforma, irá conseguir aceder ás suas transações e movimentos de saldos, ao saldo do banco e tem acesso ás características do banco, ou seja, às suas configurações, tanto de criação como de possível edição.

Numa vertente de login interno, realizado pela SIBS, esta tem acesso aos mesmos processos indicados anteriormente, mas contém, a particularidade de conseguir aceder ás informações de todos os bancos, podendo também alterar as suas características e limitações.

Uma vez que existe esta plataforma, e está em contacto direto com o cliente (bancos), os testes são realizados para garantir que o SW é entregue ao cliente com qualidade.

Para a realização deste tipo de testes, foi usada a biblioteca seleniumLibrary, uma vez que é a biblioteca utilizada para o desenvolvimento de testes web dentro da *robot framework* ("SeleniumLibrary,"). Esta biblioteca utiliza módulos de *Selenium WebDriver*, com o objetivo de servir de ponte entre a robot *framework* e o mecanismo do *selenium*, controlando assim paginas de um browser (Ramya, Sindhura, & Sagar, 2017; "SeleniumLibrary,").

Com esta biblioteca é possível identificar os elementos numa página, de modo a executar ações sobre ela, com este conjunto de ações são realizados os testes automatizados. Os testes desenvolvidos, contêm ações/funções fundamentais em termos de codificação, ações estas que são utilizadas em praticamente todos os testes desenvolvidos.

```
*** Keywords ***
Realizar Login no Portal
    [Documentation]     realizar login no portal, o valor do token é que define o tipo de login

    log to console      **** KW Start: Realizar Login no Portal ****
    # Open Browser  browser=chrome
    Open Browser     ${ipPortal}     browser=chrome
    maximize browser window
    wait until element is visible       ${loginBtnSend}
    # preparar o Token para fazer login com o user passado como parametro
    log to console      TOKEN--->${Token}
    input text          ${inputToken}           ${Token}
    click element       ${loginBtnSend}
    log to console      **** KW End: Realizar Login no Portal ****
```

Figura 16 – Portal IPS, Kw realização de Login

Esta Kw (Figura 16), é utilizada em todos os casos de testes desenvolvidos para o portal, uma vez que é necessário realizar o login para conseguir utilizar a plataforma.

Sendo uma *keyword* genérica, é possível aceder a um browser com ip's diferentes, isto é importante, uma vez que, existem vários ambientes que contém a aplicação portal, por exemplo, os testes estão a ser executados num ambiente de desenvolvimento e num ambiente de testes.

As kw's chamadas estão inseridas na biblioteca do *Selenium*, e contém as ações de:

- Abrir um browser
- Maximizar um browser
- Esperar até o elemento com o *Xpath* estar presente na página
- Introduzir texto
- Clicar em botões.

```
Selecionar Itens de Menu
    [Documentation]     Selecionar itens de menu do portal
    [Arguments]         ${item1}        ${item2}        ${item3}
    log to console      **** KW Start: Selecionar Itens de Menu ****

    log to console      ${item1} > ${item2} > ${item3}

    click element       xpath=//span[text()='${item1}']
    click element       xpath=//span[text()='${item2}']
    click element       xpath=//span[text()='${item2}']/../..//ul//li//a/span[text()='${item3}']
    log to console      **** KW End: Selecionar Itens de Menu ****
```

Figura 17 – Portal IPS, Kw seleção menus

Outra *keyword* importante (Figura 17) e utilizada em todos os casos de testes, é a de selecionar os menus e sub-menu existentes na aplicação web.

Esta *keyword* é genérica, e seleciona todos os menus e sub-menus o teste necessita, utilizando as variáveis de entrada item1, item2 e item3 que irá indicar o nome do menu/sub-menu ao qual é pretendido aceder.

```
Validar utilizador que realizou o login
    [Documentation]     Validar que tipo de utilizador fez login no portal
    [Arguments]         ${utilizador}       ${tipoUtilizador}       ${Query}=${EMPTY}

    log to console      **** KW Start: Validar utilizador que realizou o login ****
    Selecionar Aba nova do portal
    ${utilizadorPagina}=        get text        ${TextBIC}
    ##validar o tipo de utilizador na BD
    ${conexao}=     Connect to DB2 Database     ${DatabaseOP}       ${HostOP}       ${PortOP}       ${UserOP}       ${PasswordOP}
    ${sql} =        set variable if  '${Query}' == 'SIBS'       SELECT ROLE FROM FTM.PARTY_PMT WHERE BIC='${utilizador}' AND STATUS = 'ENAB'        SELECT T2_PARTICIPATION FROM FTM.PARTY_PMT_SCH
    log to console  Querie Executada: ${sql}
    ${Result}=      Execute Querie Select       ${conexao}      ${sql}
    run keyword if      ${Result} == False      fail    Não existem dados na BD referentes ao banco ${utilizador}.
    run keyword if      '${Result[0]}' != '${tipoUtilizador}'       fail    O utilizador nao é do tipo ${tipoUtilizador}.
    Disconnect_to_DB2_Database      ${conexao}
    capture page screenshot
    log to console      **** KW End: Validar utilizador que realizou o login ****
```

Figura 18 – Portal IPS, Kw validação de utilizador

Uma vez que, existem diferentes perfis de acesso á plataforma web, é necessário validar o tipo de utilizador que realizou o login, isto porque, cada perfil (sibs ou banco) visualiza e realiza diferentes ações.

Nesta *keyword* (Figura 18) para além da utilização da biblioteca do *selenium* também é utilizada a *DatabaseLibrary*, de modo a ser possível aceder e realizar ações na base de dados.

É necessário ajustar os xpath dos elementos nos testes que se destinam a executar em diferentes ambientes, uma vez que, as páginas de cada plataforma web contém diferenças.

Para ajustar esse *xpath*, forma criados ficheiros auxiliares (Figura 19) que incluem os *xpath's* de forma diferenciada por ambiente, com isto garantimos que cada execução utiliza os *xpath* associados à plataforma, garantindo também a alteração de *xpath* numa determinada página, assim o *tester* só necessita de realizar alterações num ficheiro, em vez de alterar todos os casos de testes que contenham ações sobre o elemento que sofreu alterações.

Com a adição de *xpaths*, num ficheiro separado do teste, caso pretenda usar um *xpath*, apenas é necessário colocar a variável na chamada do elemento (Click element $(BtmVoltar)).

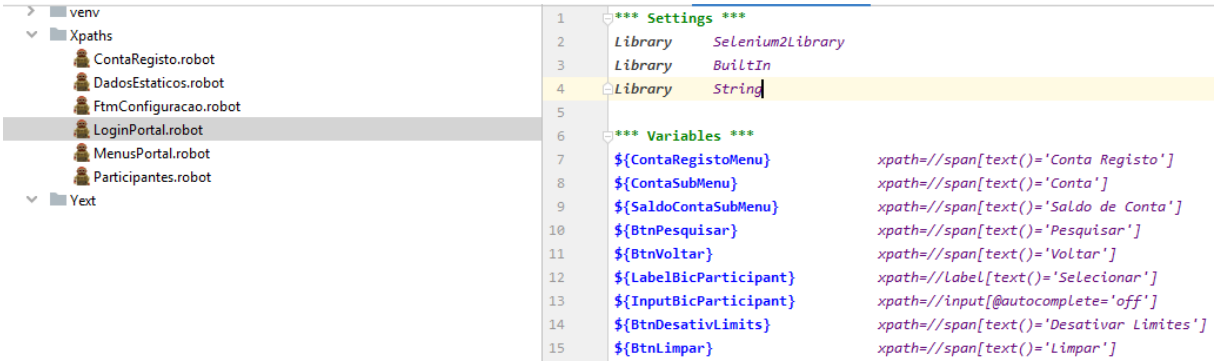


Figura 19 – Ficheiro auxiliar de Xpath's

3.2.4.5 Testes ao *Cognos*

Cognos, é uma ferramenta utilizada para a criação de relatórios baseada na web. Ela é integrada no portal Web do IPS e está disponível à SIBS e aos Bancos de forma a poderem extrair relatórios em formato PDF ou formato CSV, assim como visualizar alguns gráficos de transações. Esta ferramenta permite exportar as pesquisas de transações e movimentos que são apresentados no portal e integrar nos seus sistemas para análise.

Existe uma aplicação java que permite por linha de comandos aceder à interface do *cognos* e efetuar um pedido de geração de relatório ou gráfico ao *cognos* e obter o ficheiro resultante. Esta aplicação (ficheiro jar) foi integrado na robot *framework*, via *python* usando o módulo subprocess e permite obter alguns indicadores, nomeadamente o tempo de execução de extração de um relatório para efeitos de performance; validar que o relatório existe no sistema; e que o mesmo executa sem erro. Para obter estes indicadores é processado o output da aplicação java e é com base nesse output que é tomada a decisão de sucesso ou insucesso do teste.

3.2.4.6 Testes a funcionalidades do IPS recorrendo á base de dados

Dentro da equipa do IPS também existem processos de base de dados, neste caso a base de dados utilizada é em DB2[11].

O desenvolvimento deste tipo de testes, é sobretudo realizado com base nas necessidades que vão surgindo, uma vez que, os outros tipos de testes mencionados anteriormente já possuem *keywords* que realizam validações á base de dados, estando assim a testar diversos processos.

Desde o início da implementação deste tipo de testes, foi necessário desenvolver um teste para validar a inexistência de incoerência de dados, ou seja, foi necessário validar o somatório de todos os saldos disponíveis, e suas reservas (saldo que fica cativo no momento em que se inicia a transação financeira, garantindo que quando a transação é finalizada, existe saldo para realizar a transação com sucesso, após esta finalização o saldo cativo deixa de ficar no banco que realiza a transação, sendo adicionado ao banco que recebe a transação).

Este somatório é referente ao dia atual e ao dia anterior, validando que ambos os somatórios dão o mesmo resultado. Caso esta regra não se aplique, teremos problemas graves, pois, mostra que existiu dinheiro que desapareceu, ou apareceu na conta de alguém sem justificação lógica.

Para a realização de testes a base de dados, foi necessário o auxílio do *python*, uma vez que é melhor no que diz respeito á execução de comandos sql e tratamentos dos seus dados, algo que é importante no teste de validação de saldos. Dentro do *python* foi utilizada a biblioteca ibm_db, para realizar a conexão e executar processos de db2.

3.2.5 Execução de testes

Os casos de teste foram integrados no *jenkins*, com o objetivo de estes executarem diariamente de forma automática. Para isso foi necessário a criação e configuração de jobs, e por fim a configuração de baterias de execução diária.

O *jenkins* está organizado por aplicação (Base de dados, Cognos, FTM, Portal e Sistema), de forma a serem organizados os casos de testes, sendo essencial caso alguém queira aceder a um caso de teste de forma mais rápida. Dentro de cada caso de teste, também

[11] Sistema de Gestão de Base de dados Relacionais produzido pela IBM.

foi definida uma sub organização por ambiente (DEV[12], TST[13], TST VS[14]), facilitando quando existe a necessidade de escolher o ambiente para o qual queremos a execução do teste.

Foram definidas variáveis de ambiente no *jenkins*, de modo a indicar o caminho do projeto robot *framework* onde estão desenvolvidos os casos de teste, e definir também o *workspace* sendo o espaço de trabalho nas execuções dos testes. Este tipo de variáveis contribuem para a gestão de possíveis alterações, pois, caso seja necessário alterar o caminho de algumas destas variáveis, apenas realizamos a alteração num local, em vez de alterar cada caso de testes.

Sendo um projeto interno da equipa IPS, foi necessário realizar a atribuição de utilizadores de login, para que todos os elementos da equipa tenham acesso aos casos de teste, e á parte administrativa do jenkins.

3.2.5.1 Criação e configuração de jobs

Para criar um job no jenkins é necessário recorrer ao “New Item” -> ”Freestyle Project”, este tipo de projeto, abrange várias operações, permitindo que seja possível especificar a chamada de testes, tendo como objetivo este tipo de projetos adaptar tarefas simples dentro de um projeto.

Após criação deste tipo de projeto, é necessário configurá-lo:

- Adicionar as variáveis de entrada do teste;
- Definir estratégias para execuções antigas, limitando o máximo de execuções guardadas e o número de dias para guardar as execuções;
- Restringir onde o teste rá ser executado, pode ser necessário executar um caso de teste numa máquina que contenha configurações diferenciadas;
- Definição do URL de repositório onde se encontra o teste desenvolvido, neste caso configuração do GIT;
- Indicação do *workspace* onde estará o teste após execução, como cada caso de teste terá a sua área diferenciada, foi configurado com a variável de ambiente do workspace, concatenando com o nome do caso de teste. Cada *workpace* contém dados da última execução do teste, e após início de uma nova execução os dados são apagados, e guarda a informação da nova execução.

[12] Ambiente de desenvolvimento
[13] Ambiente de testes apontada para a máquina de Lisboa
[14] Ambiente de testes apontada para a máquina de Viseu

- Definição da pasta que irá ter o relatório HTML da execução de teste, neste caso fica em cada *workspace* tendo uma pasta denominada de "*Result*".
- Criação de um comando *batch*, com o objetivo de execução do script de teste. Este comando contém a indicação, da localização onde se encontra o caso de teste (*.robot*), e passagem de variáveis de entrada, necessárias para a execução do mesmo.

A Figura 20 representa as configurações de variáveis e do comando batch que existem nas builds do jenkins.

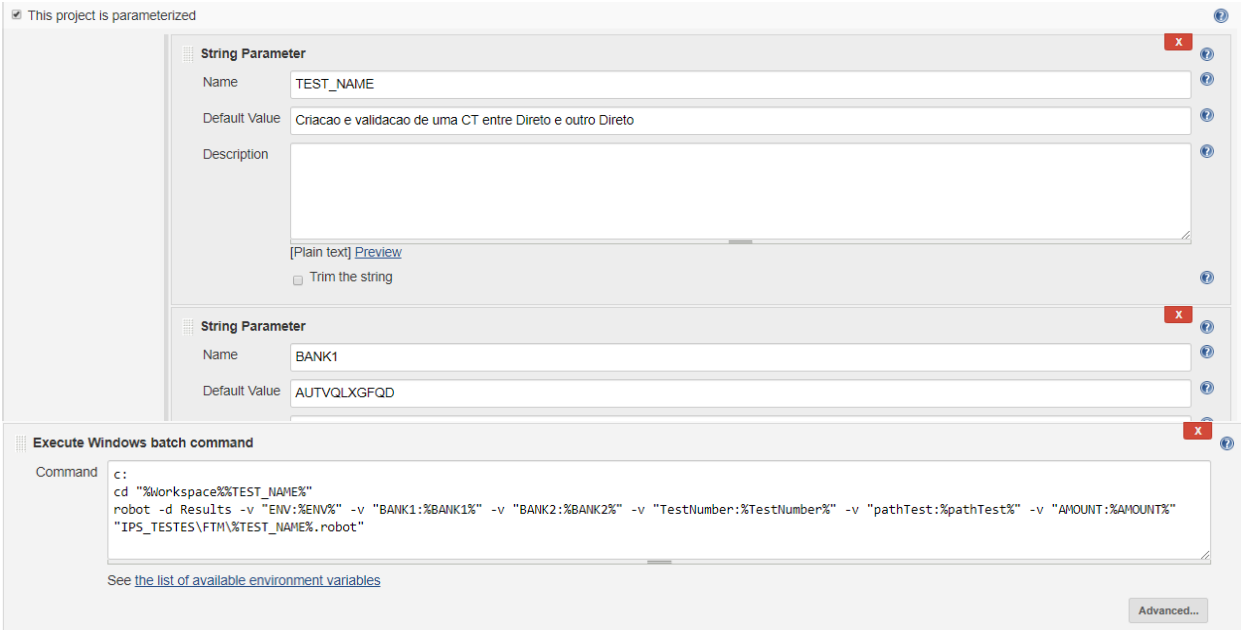


Figura 20 – Configurações builds Jenkins

3.2.5.2 Configuração de baterias de regressão

As baterias de regressão criadas no *jenkins*, têm como objetivo a recolha de diversos casos de teste, para que estes sejam executados de forma automática. Para além das baterias de execução diária, também está estruturada baterias de teste que podem ser executadas a qualquer momento, basta existir uma ação manual sobre elas.

Para criação de uma bateria de execução de testes, é criado um *Multijob project*, com este tipo de projeto é possível adicionar diversos jobs de modo a serem executados em sequência ou em paralelo.

No caso deste projeto, as baterias de testes foram configuradas da seguinte forma:

- Definir estratégias para execuções antigas, limitando o máximo de execuções guardadas e o número de dias para guardar as execuções;
- Caso seja uma bateria de execução automática, é necessário realizar o agendamento, "Build Periodically", ajuda nesse ponto, sendo possível agendamentos por hora, de x em x horas, todos os dias, dias específicos.
- Adicionar a chamada dos jobs criados, que sejam para fazer parte da bateria de testes que estamos a configurar, neste caso, para além de identificar os jobs também é possível definir ações sobre cada um. Existindo um caso de teste para validação do ambiente, caso falhe não faz sentido executar os restantes, pois, o ambiente não está em condições para a realização de testes, tudo isto é configurável, na secção "MultiJob Phase", como mostra a Figura 21.

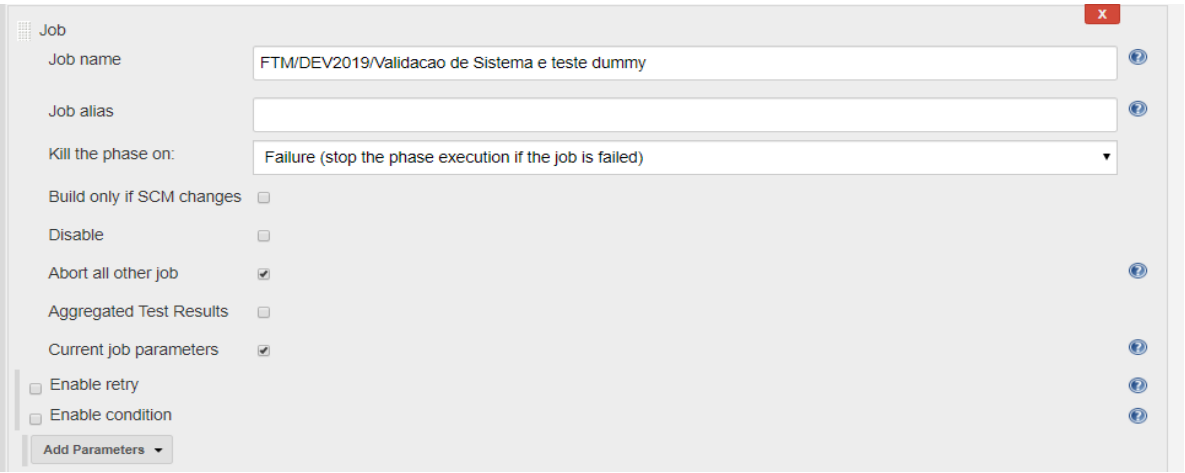


Figura 21 – Jenkins MultiJob Phase

3.2.6 Benefícios e conclusões do caso de estudo

Uma das formas de manter a qualidade do projeto, é a realização de testes manuais e automatizados. Existem vários pontos, que valorizam este projeto, dentro da equipa do IPS.

Com a execução dos casos de testes permitiu que cada entrega de software garantisse os requisitos mínimos de qualidade ligados á importância da não existência de quebras no serviço. Para além disso, é importante na vertente da confiança da equipa, quando existem entregar de software.

Outra problemática era o esforço exigido a cada elemento da equipa de forma a conseguir testar as funcionalidades desenvolvidas e/ou correções, incluindo todas as suas dependências. Poderia ter múltiplas dependências e as mesmas terem uma ordem de execução definida e devido à complexidade e por vezes exigir algum tempo, levando a que por vezes apenas se realize pequenos testes unitários em vez de testar toda a sequência de passos até testar efetivamente o desenvolvimento que foi feito. Com a definição de testes automáticos que garantem a execução das dependências, não só valida que nenhuma funcionalidade foi afetada como garante que o desenvolvimento é testado em condições reais, sem adulteração de dados.

Realizando uma cobertura de testes às funcionalidades core do sistema permite garantir que não existe regressão, evitando que o software perca funcionalidades quando o mesmo é entregue ao cliente.

Com os testes a executarem diariamente, conseguimos validar o comportamento das aplicações do IPS, uma vez que as etapas do teste desenvolvido incluem todas as validações do comportamento normal do SW, se há algo que está diferente e faz falhar o teste, este vai ser objeto de uma análise mais profunda, de modo a ser possível agir de forma mais rápida e corrigir o problema encontrado. Com a automação de testes também existe uma maior rapidez na replicação de problemas, ou seja, qualquer pessoa da equipa pode executar um caso de teste a qualquer momento a partir do *jenkins*, retirando evidencias de forma mais rápida, exportando as evidências em HTML existentes em cada execução de teste.

Com a execução diária de testes, conseguimos introduzir volumetria na nossa base de dados, isto é algo fundamental, devido ás questões de performance, validando assim a otimização das *queries* DB2 usadas para obtenção de dados.

As ferramentas encontradas para solucionar o problema, contribuíram para uma metodologia de trabalho, permitindo encontrar erros no software. Um caso prático onde os testes automatizados se mostraram essenciais aconteceu na validação dos dados apresentados numa página da componente Web de pesquisa de movimentos financeiros. Esta página apresenta os movimentos financeiros e o saldo após uma determinada operação e havia necessidade de avaliar dezenas de registos ao longo dos dias e validar que, todas as operações estavam a ser apresentadas e os saldos estavam corretos. A Figura 22 representa os dados iniciais que se encontravam no sistema, algo que não daria para realizar a validação pretendida.

Não era viável ter um recurso humano, dedicado a realizar dezenas de transações ao longo de vários dias e então recorreu-se à automação de testes para realizar uma bateria de testes com transações com espaçamento de uma hora entre elas. Desta forma, a pesquisa retornou todos os dados necessários para avaliar a conclusão do teste como é visível na Figura 23 sendo testada a performance da pesquisa devido à volumetria criada. Assim, a automatização de testes permitiu libertar um recurso da equipa, para realizar outras tarefas do projeto e criou-se volumetria de dados essenciais para analisar e detetar futuros problemas.

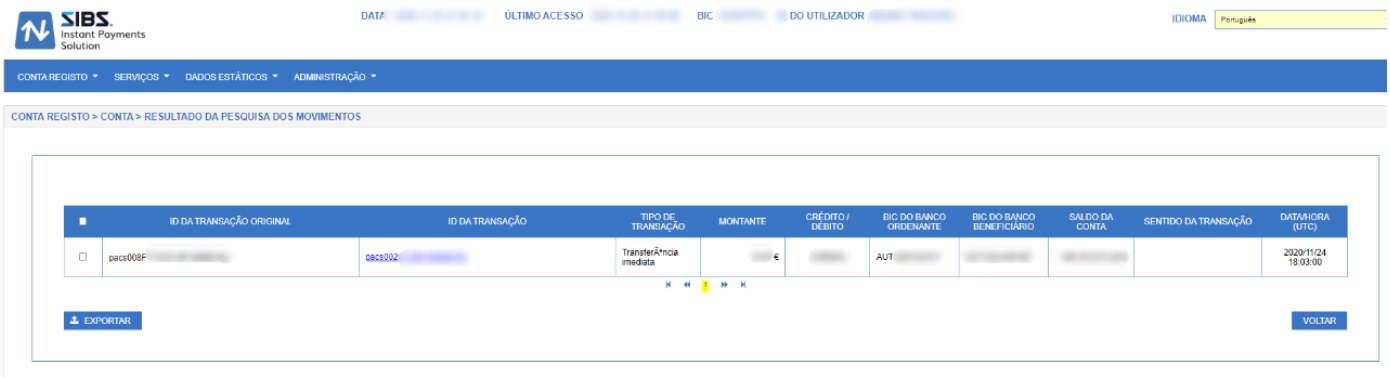


Figura 22 - Dados iniciais no sistema

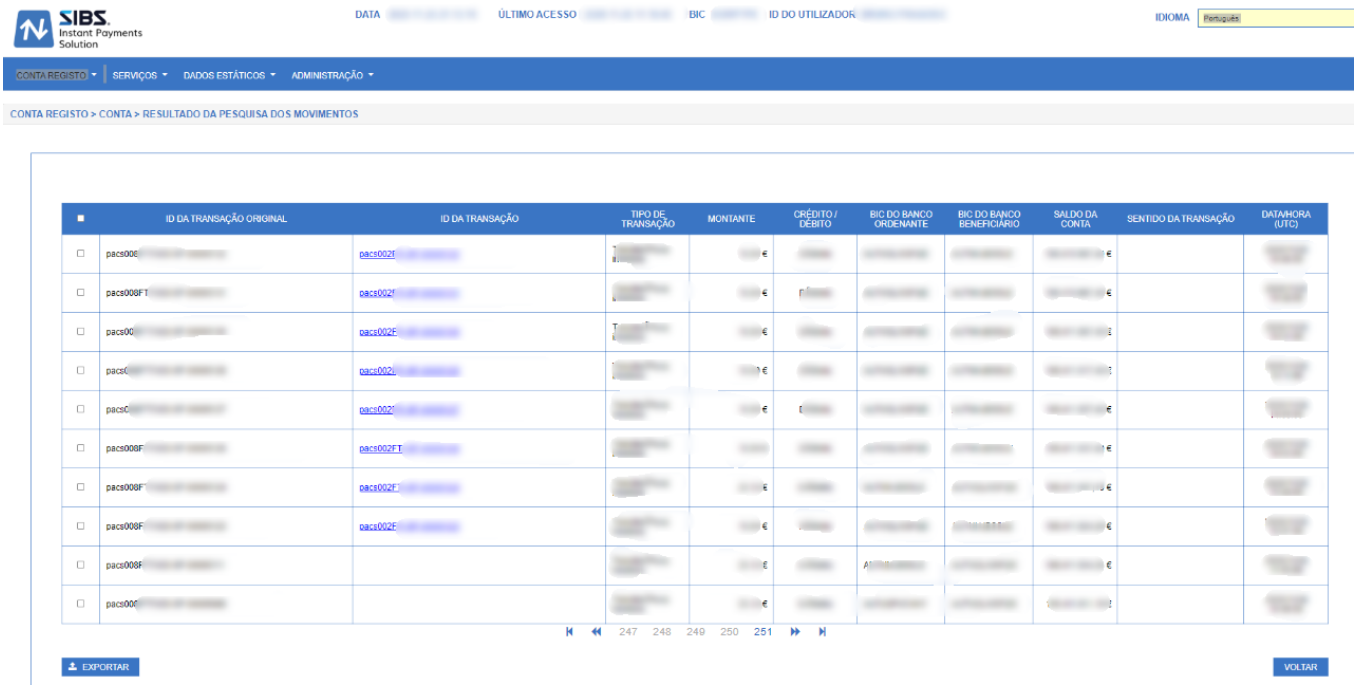


Figura 23 - Dados finais no sistema

Um outro caso detetado pela automatização de testes, que permitiu evitar que fosse lançada uma versão de software, com um erro num relatório *Cognos*. Por engano, tinha sido renomeado um dos relatórios e que deixara de funcionar no portal, pois, deixou de ser encontrado e iria dar um erro ao tentar obter o relatório. Este caso passou despercebido e apenas foi detetado, pelos testes automáticos realizados e que deram conta de uma exceção, na sua execução. Isto permitiu, evitar que fosse introduzido um erro de regressão e levar a uma reclamação posterior de um cliente.

A Figura 24 representa a evolução de defeitos existentes no software, com e sem testes. Esta evolução representa o que está a acontecer na equipa do IPS, sendo que o gráfico foi realizado com dados reais, mas devido à política confidencialidade aplicada na empresa, estes não poderão ser divulgados.

Com os dados recolhidos até novembro de 2020, conseguimos observar uma diminuição de defeitos, como é possível analisar a meio da linha temporal do gráfico presente na Figura 24. Estes mesmos dados dão-nos uma antecipação de um cenário de erros nos próximos meses no que concerne ao número de casos de teste e por conseguinte uma redução de defeitos no software. Com este incremento de casos de testes, as possíveis falhas no software são identificadas prematuramente, evitando assim, falhas nos processos de certificação e qualidade pelos quais os softwares serão sujeitos em cada entrega.

Apesar de se observar uma notória diminuição de falhas, esta nunca será nula, mas irá permitir uma maior agilização no processo de gestão dos defeitos em cada período de desenvolvimento, alocando os recursos humanos no desenvolvimento de novas funcionalidades e não apenas na correção de defeitos

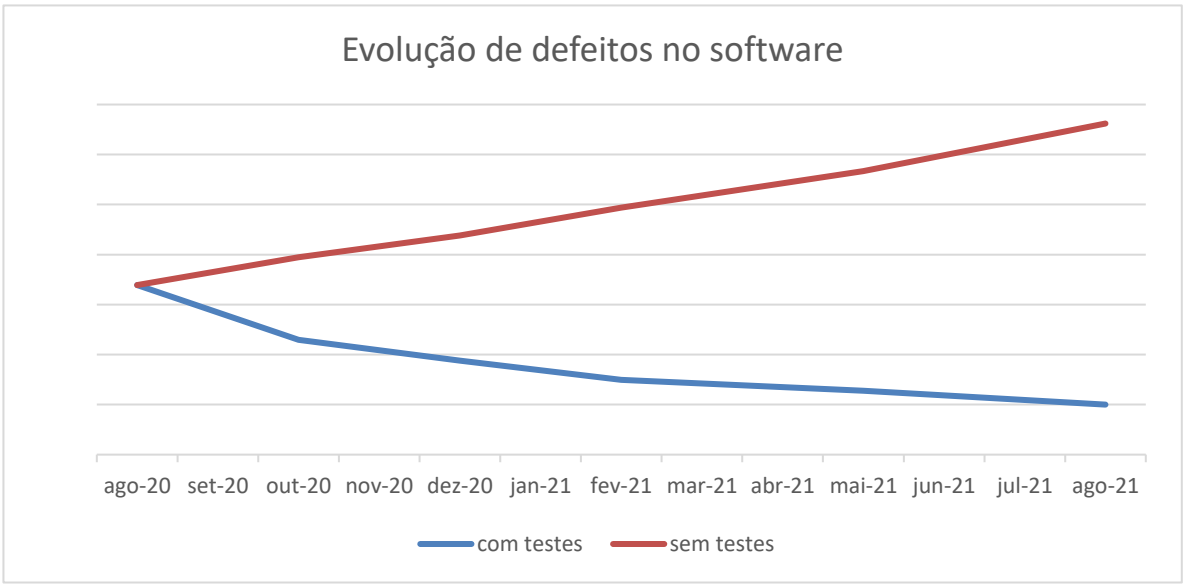


Figura 24 - Evolução de defeitos no software

4. Conclusão

Os testes de software são criados para encontrar falhas no sistema, numa visão em que um teste com sucesso é aquele que encontra falhas. Estes contemplam atividades padrão, das quais devem ser criados requisitos, para que seja possível fazer a comparação de valores reais, com os valores dos scripts de execução.

Podemos questionar, porque acontecem falhas no sistema? Se os projetos de desenvolvimento são definidos e estruturados? Isto tudo pode acontecer devido ao comportamento humano. O ser humano comete um erro, este erro origina um defeito no sistema, e por sua vez, com a execução do sistema é visível a falha, originada pelo defeito. Pelos motivos mencionados anteriormente, os testes são a fase mais crítica no desenvolvimento de software. Para além da fase de testes, também é possível projetar e realizar testes em todas os estágios de desenvolvimento, tendo em conta que cada etapa de testes, acompanha cada fase de desenvolvimento, potenciando a execução de testes em cada uma, corrigindo as ações que poderão conter erros.

Relativamente a testes de software, podemos direcioná-los em duas vertentes, manuais e automatizados. Os testes manuais, como o próprio nome indica, são executados de forma manual por um ser humano, asseverando-se mais eficaz em projetos pequenos ou que se encontrem em constante mudança, dado que encontram problemas que podem ser provocados pelo comportamento humano. Quanto aos testes automatizados, nestes são usadas ferramentas

e scripts de execução, sendo mais eficazes em grandes projetos, em projetos rotineiros, ou em processos bem definidos. Com as definições de tipos de testes, podemos afirmar que os testes manuais e automatizados se complementam, sendo importante contemplar estes dois tipos de testes num projeto de software. A integração da automação de testes é sempre uma mais valia em qualquer projeto.

Hoje em dia, existem muitos riscos associados à qualidade dos projetos, tais como:

- A rotação das pessoas em novos projetos/novas empresas, que acaba por ser algo recorrente;
- Complexidade das áreas de negócios;
- Mudanças de requisitos a meio do projeto;
- Centralização de código que é uma boa prática, contudo, por vezes podemos estar a alterar código que tem impacto sem nos lembrarmos das consequências, e quanto mais júnior for a pessoa mais riscos terá.

Uma das formas de manter a qualidade do projeto em detrimento destes e muitos mais riscos, é a componente dos testes. As ferramentas de automação de testes são utilizadas para executar testes com base em algoritmos, comparando assim o resultado do script com o resultado pretendido, afigurando-se uma ajuda no que diz respeito à reutilização de *scripts*.

Existem vários tipos de ferramentas para cada categoria, as ferramentas de testes unitários validam cada unidade de forma isolada, as ferramentas de testes funcionais são utilizadas para garantir que os requisitos são os esperados, ferramentas de análise de cobertura de testes servem para medir quais as funcionalidades de software estão cobertas pela automação de testes, ferramentas de gestão de testes guardam informações acerca dos testes e as ferramentas de performance servem para validar o desempenho do componente de software, ou mesmo como um todo.

Em termos práticos, foram avaliadas quatro ferramentas de automação de testes, *Katalon Studio, Robot Framework, Protractor, Watir*, de modo a compreender qual se enquadrava melhor nos desenvolvimentos do IPS, sendo selecionada a robot framework, devido ao seu tempo de compilação e execução de testes, o que contribui para o desempenho dos testes, o que leva a ser possível testar mais em menos tempo. Para além dos pontos referidos, a *robot framework* é de fácil compreensão e a sua documentação é clara, sendo uma vantagem para quem vai desenvolver os testes.

Ocorreram dificuldades durante a instalação de ferramentas e bibliotecas, uma vez que não existe acesso à rede exterior. Esta situação levou à realização de download de todos os componentes e suas dependências, tendo sido necessário passar os ficheiros para as máquinas da SIBS.

A especificação de testes é uma fase essencial num projeto, pois é com base nesta especificação que os testes são elaborados, tendo indicações das ações que o teste deve ter, requisitos para a sua execução, e o resultado esperado do teste. Os testes que se encontram na especificação foram analisados inicialmente, posteriormente desenvolvidos, e estão a ser executados diariamente, com o auxílio do *jenkins*. Após a implementação, verificou-se mais confiança na equipa, aumento da qualidade do SW, redução da carga de trabalho que existia nas pessoas, pois alguns casos de testes podem ser executados via automação. Com as execuções diárias e execuções antes da entrega de software, conseguimos agir de forma mais rápida quando existe algum comportamento que não é o esperado.

Os casos de testes referenciados no projeto, foram implementados na SIBS com sucesso, sendo que os resultados atuais são promissores, pois permitem a identificação de um maior número de erros de forma prematura, o que leva a uma rapidez na resolução dos problemas, originando uma maior qualidade no software.

Sem dúvida alguma, a componente dos testes é essencial para manter a qualidade de um software.

4.1 Trabalho Futuro

Como trabalho futuro, propõe-se adicionar uma melhoria no tipo de testes a realizar. No IPS para comunicar com os bancos é necessário recorrer a sistemas externos, sendo importante a realização de testes E2E, estes testes testam fluxos do início ao fim, validando o processo em todos os sistemas que ele passa para a realização de uma funcionalidade.

Num futuro próximo, o IPS irá fazer upgrade das tecnologias utilizadas, e nessa altura de mudança está previsto existirem mais casos de testes, para termos mais cobertura de modo a serem validadas as funcionalidades, verificando quais sofreram alterações com o upgrade das tecnologias.

Para que seja possível, termos uma perceção dos resultados das execuções dos testes sem a necessidade de realizar login no *jenkins* vai ser realizado um script em *python* para

enviar o relatório diário via e-mail para toda a equipa. Este ponto era algo que estava programado no desenvolvimento deste projeto mas devido ao COVID-19 a parte prática esteve parada alguns meses dada a impossibilidade de acesso físico às instalações da SIBS, para se poder transferir bibliotecas de instalação para a máquina, dado que a mesma não tem acesso à rede exterior.

Outro aspeto importante a desenvolver, são as métricas de tempos médios das transações de modo a verificar possíveis deterioramentos e *queries* mal implementadas. Mensalmente vai ser realizado um quadro resumo para efeitos visuais rápidos, onde é possível observar os testes realizados, as médias de falhas/sucessos e tempos de execução, para realizarmos uma análise, do que está a correr bem e mal nos casos de testes, para vermos como melhorar.

REFERÊNCIAS

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.

Aebersold, K. What is a Test Framework? , from https://smartbear.com/learn/automated-testing/test-automation-frameworks/

Afzal, W. (2007). Metrics in software test planning and test design processes.

Agarwal, B., Tayal, S., & Gupta, M. (2010). *Software engineering and testing*: Jones & Bartlett Learning.

Altvater, A. (2017). Stackify, 'Code Coverage Tools: 25 Tools for Testing in C, C++, Java'. Retrieved 09-06-2020

Ammann, P., & Offutt, J. (2016). *Introduction to software testing*: Cambridge University Press.

Andrea, J. (2007). Envisioning the next-generation of functional testing tools. *IEEE Software, 24*(3), 58-66.

angularjs. Using Protractor. Retrieved 17-09-2020, 2020, from https://docs.angularjs.org/guide/e2e-testing

Atlassian. Atlassian, 'About Code Coverage - Atlassian Documentation. Retrieved 10-06-2020

Audy, J. (2015). *Scrum 360: Um guia completo e prático de agilidade*: Editora Casa do Código.

Bajaj, H. (2015). Choosing the right automation tool and framework is critical to project success. *Infosys Limited, 41*.

Bandwal, S., Choudhary, A., Kulkarni, S., Bhase, K., Shahani, S., & Pawar, A. (2019). *Automation Framework for testing Dynamic Configurable tool.* Paper presented at the 2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA).

Barna, C., Litoiu, M., & Ghanbari, H. (2011). *Autonomic load-testing framework.* Paper presented at the Proceedings of the 8th ACM international conference on Autonomic computing.

Base36. (21 Mar. 2015). Automated Vs. Manual Testing: The Pros And Cons Of Each. *N.p., 2013. Web. 21 Mar.*

Bellatrix. (2019). Generations of Test Automation Frameworks. from https://bellatrix.solutions/generations-of-test-automation-frameworks-past-and-future/

Bhatt, D. (2017). A Survey of Effective and Efficient Software Testing Technique and Analysis. *Iconic Research and Engineering Journals (IREJOURNALS)*.

Bisht, S. (2013). *Robot framework test automation*: Packt Publishing Ltd.

Craig, R. D., & Jaskiel, S. P. (2002). *Systematic software testing*: Artech house.

Crosby, P. B. (1979). *Quality is free: The art of making quality certain* (Vol. 94): McGraw-hill New York.

De Grood, D.-J. (2008). Test Automation. *TestGoal: Result-Driven Testing*, 277-292.

de Mello Cordeiro, J. V. B. (2004). Reflexões sobre a Gestão da Qualidade Total: fim de mais um modismo ou incorporação do conceito por meio de novas ferramentas de gestão? *Revista da FAE, 7*(1).

Fat, N., Vujovic, M., Papp, I., & Novak, S. (2016). *Comparison of AngularJS framework testing tools.* Paper presented at the 2016 Zooming Innovation in Consumer Electronics International Conference (ZINC).

Financial Transaction Manager. from https://www.ibm.com/support/knowledgecenter/SSRH32_3.2.4/fxhovr.html

Fowler, M., & Foemmel, M. (2006). Continuous integration.

Framework, R. (2020). Robot Framework Introduction. Retrieved 18-9-2020, 2020, from https://robotframework.org/#introduction

Garousi, V., & Elberzhager, F. (2017). Test automation: not just for test execution. *IEEE Software, 34*(2), 90-96.

Gogna, N. (2014). Study of browser based automated test tools watir and selenium. *International Journal of Information and Education Technology, 4*(4), 336.

Gogna, N., & Kumari, R. (2011). Comparative study of browser based open source testing tools watir and wet. *International Journal on Computer Science and Engineering, 3*(5), 1910-1923.

Graham, D., Van Veenendaal, E., & Evans, I. (2008). *Foundations of software testing: ISTQB certification*: Cengage Learning EMEA.

Harsha, T., & Kumari, B. S. Software Test Automation with Robot Framework. *International Journal on Recent and Innovation Trends in Computing and Communication, 5*(6), 432-433.

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer, 34*(9), 120-127.

Homès, B. (2013). *Fundamentals of software testing*: John Wiley & Sons.

Hooda, I., & Chhillar, R. S. (2015). Software test process, testing types and techniques. *International Journal of Computer Applications, 111*(13).

Huizinga, D., & Kolawa, A. (2007). *Automated defect prevention: best practices in software management*: John Wiley & Sons.

Hussain, S., Wang, Z., Toure, I. K., & Diop, A. (2013). Web service testing tools: A comparative study. *arXiv preprint arXiv:1306.4063*.

Islam, N. (2016). A Comparative Study of Automated Software Testing Tools.

ISTQB. (2011). *ISTQB Certified Tester Foundation Level, Study Guide (2011 Syllabus)*

Study Guide version 2011.5.

J. Itkonen, M. V. M. a. C. L. (2009). How do testers do it? An exploratory study on manual testing practices. *2009 3rd International Symposium on Empirical Software Engineering and Measurement, Lake Buena Vista, FL, 2009, pp. 494-497.*

Kan, S. H. (2002). *Metrics and models in software quality engineering*: Addison-Wesley Longman Publishing Co., Inc.

katalon. A Comparison of Automated Testing Tools. Retrieved 16-09-2020, 2020, from https://www.katalon.com/resources-center/blog/comparison-automated-testing-tools/

Klaus Olsen (chair), M. P., Stephanie Ulrich. (2018). Certified tester foundation level syllabus. *International Software Testing Qualifications Board*.

Ko, A. J., & Myers, B. A. (2005). A framework and methodology for studying the causes of software errors in programming systems. *Journal of Visual Languages & Computing, 16*(1-2), 41-84.

Larman, C. (2004). *Agile and iterative development: a manager's guide*: Addison-Wesley Professional.

Layton, M. C. (2015). *Scrum for dummies*: John Wiley & Sons.

Lazic, L. M., M. (2020). SOFTWARE QUALITY ENGINEERING versus SOFTWARE TESTING PROCESS. *SIEMENS d.o.o, Radoja Dakića 7, 11070 Beograd, Serbia&Montenegro*.

Levy, Y., & Ellis, T. J. (2006). A systems approach to conduct an effective literature review in support of information systems research. *Informing Science, 9*.

Lewis, W. E. (2017). *Software testing and continuous quality improvement*: Auerbach publications.

Mailewa, A., Herath, J., & Herath, S. (2015). *A Survey of Effective and Efficient Software Testing.* Paper presented at the The Midwest Instruction and Computing Symposium. Retrieved from http://www. micsymposium. org/mics2015/ProceedingsMICS_2015/Mailewa_ 2D1_41. pdf.

Milad Hanna , A. E. A. a. M. M. M. (2018). *Automated Software Testing Framework for Web Applications*. International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 11 (2018) pp. 9758-9767.

Na, Q., & Huaichang, D. (2015). *Extension and application based on robot testing framework.* Paper presented at the 2015 6th IEEE International Conference on Software Engineering and Service Science (ICSESS).

Olan, M. (2003). Unit testing: test early, test often. *Journal of Computing Sciences in Colleges, 19*(2), 319-328.

Pajunen, T., Takala, T., & Katara, M. (2011). *Model-based testing with a general purpose keyword-driven test automation framework.* Paper presented at the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops.

Pereira, M. G., & Galvão, T. F. (2014). Etapas de busca e seleção de artigos em revisões sistemáticas da literatura. *Epidemiologia e Serviços de Saúde, 23*, 369-371.

Prof . Swati Dubey, P. S. T., Prof .Dipti Dighe. (2017). *STUDY ON VARIOUS PHASES OF SOFTWARE TESTING LIFE CYCLE*. Paper presented at the 7th International Conference on Recent Trends in Engineering, Science & Management, Genba Sopanrao Moze College Of Engineering, Balewadi-Baner, Pune.

Protractor. Protractor End to End testing for angular. Retrieved 17-09-2020, 2020, from https://www.protractortest.org/#/

Purushothaman, S. K., Kashyap, N., Divya, H., Agarwal, S., Iqbal, S., & Behere, V. (2018). *Unified Approach Towards Automation of any Desktop Web, Mobile Web, Android, iOS, REST and SOAP API Use Cases.* Paper presented at the 2018 International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET).

Ramya, P., Sindhura, V., & Sagar, P. V. (2017). *Testing using selenium web driver.* Paper presented at the 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT).

Rierson, L. (2017). *Developing safety-critical software: a practical guide for aviation software and DO-178C compliance*: CRC Press.

Rubin, K. S. (2012). *Essential Scrum: A practical guide to the most popular Agile process*: Addison-Wesley.

Sampaio, R. F., & Mancini, M. C. (2007). Estudos de revisão sistemática: um guia para síntese criteriosa da evidência científica. *Brazilian Journal of Physical Therapy, 11*(1), 83-89.

Schwaber, C., & Gilpin, M. (2005). Evaluating automated functional testing tools. *Forrester Research*.

SeleniumLibrary. Retrieved 06-10-2020, 2020, from https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html

Seth, N., & Khare, R. (2015). *ACI (automated Continuous Integration) using Jenkins: Key for successful embedded Software development.* Paper presented at the 2015 2nd International Conference on Recent Advances in Engineering & Computational Sciences (RAECS).

Sharma, L. Why is Testing Necessary? Retrieved 13-10-2020, 2020, from https://www.toolsqa.com/software-testing/istqb/why-is-testing-necessary/

SIBS. Retrieved 10-01-2020, 2020, from https://www.sibs.com/empresa/

Sims, C., & Johnson, H. L. (2012). *Scrum: a breathtakingly brief and agile introduction*: Dymax.

Singh, I., & Tarika, B. (2014). Comparative analysis of open source automated software testing tools: Selenium, sikuli and watir. *International Journal of Information & Computation Technology, 4*(15), 1507-1518.

Smart, J. F. (2011). *Jenkins: The Definitive Guide: Continuous Integration for the Masses*: " O'Reilly Media, Inc.".

Stocks, P., & Carrington, D. (1993). *Test template framework: A specification-based testing case study.* Paper presented at the Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis.

Swati. (2020). What Is Software Testing Life Cycle (STLC)? Retrieved 05-10-2020, 2020, from https://www.softwaretestinghelp.com/what-is-software-testing-life-cycle-stlc/

Testim. What Is the Software Testing Life Cycle? Retrieved 05-10-2020, 2020, from https://www.testim.io/blog/software-testing-life-cycle/

Uddin, A., & Anand, A. (2019). Importance of Software Testing in the Process of Software Development. *IJSRD-International J. Sci. Res. Dev., 6*, 2321-0613.

Umar, M. A., & Zhanfang, C. (2019). A Study of Automated Software Testing: Automation Tools and Frameworks.

Umar, M. A. a. C., Zhanfang. (2019). A Study of Automated Software Testing: Automation Tools and Frameworks.

Vila, E., Novakova, G., & Todorova, D. (2017). *Automation Testing Framework for Web Applications with Selenium WebDriver: Opportunities and Threats*. Bangkok, Thailand: Association for Computing Machinery.

Referências

ANEXO 1

Inicio do Projeto	10-8-19		
Categoria	**Tarefa**	**Inicio**	**Fim**
Proposta Inicial	Seleção tema da tese	08-10-2019	13-10-2019
	Criação Doc. Contextualização e objetivos	05-11-2019	07-11-2019
	Leitura inicial do estado da arte	20-11-2019	29-11-2019
	Revisão do doc. Proposta da tese	01-12-2019	02-12-2019
Proposta Final	Aprofundar conceitos do estado da arte	02-12-2019	04-12-2019
	Apresentação da proposta	14-12-2019	14-12-2019
	Entrega da proposta a conselho técnico científico	17-12-2019	04-01-2020
Desenvolvimento do projeto	Escolha da metodologia de desenvolvimento	20-01-2020	25-01-2020
	Revisão do estado da arte	01-02-2020	10-11-2020
	Análise de casos de testes a implementar	01-03-2020	10-03-2020
	Análise dos requisitos	10-03-2020	20-03-2020
	Análise das ferramentas de automação de testes	21-03-2020	01-04-2020
	Instalação de ferramentas e bibliotecas	05-04-2020	08-04-2020
	Especificação dos casos de testes	12-04-2020	01-05-2020
	Criação de casos de testes	01-07-2020	24-09-2020
	Criação de baterias de testes	10-08-2020	24-09-2020
Entrega Final	Revisão do documento e entrega	11-11-2020	20-11-2020
	entrega do projeto	02-12-2020	02-12-2020

ANEXO 1 – Cronograma Detalhado

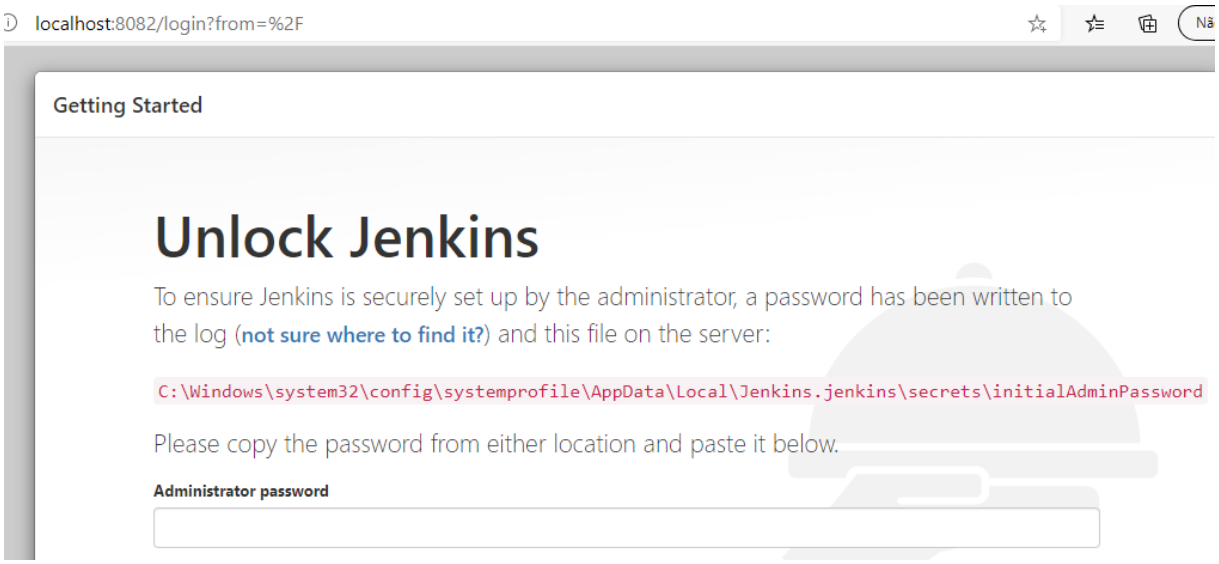
ANEXO 1


10-8-19
11-27-19
1-16-20
3-6-20
4-25-20
6-14-20
8-3-20
9-22-20
11-11-20
12-31-20
Proposta Inicial
Seleção tema da tese
Criação Doc. Contextualização e objetivos
Leitura inicial do estado da arte
Revisão do doc. Proposta da tese
Proposta Final
Aprofundar conceitos do estado da arte
Apresentação da proposta
Entrega da proposta a conselho técnico cientifico
Desenvolvimento do projeto
Escolha da metodologia de desenvolvimento
Revisão do estado da arte
Análise de casos de testes a implementar
Análise dos requisitos
Análise das ferramentas de automação de testes
Instalação de ferramentas e bibliotecas
Especificação dos casos de testes
Criação de casos de testes
Criação de baterias de testes
Entrega Final
Revisão do documento e entrega
entrega do projeto

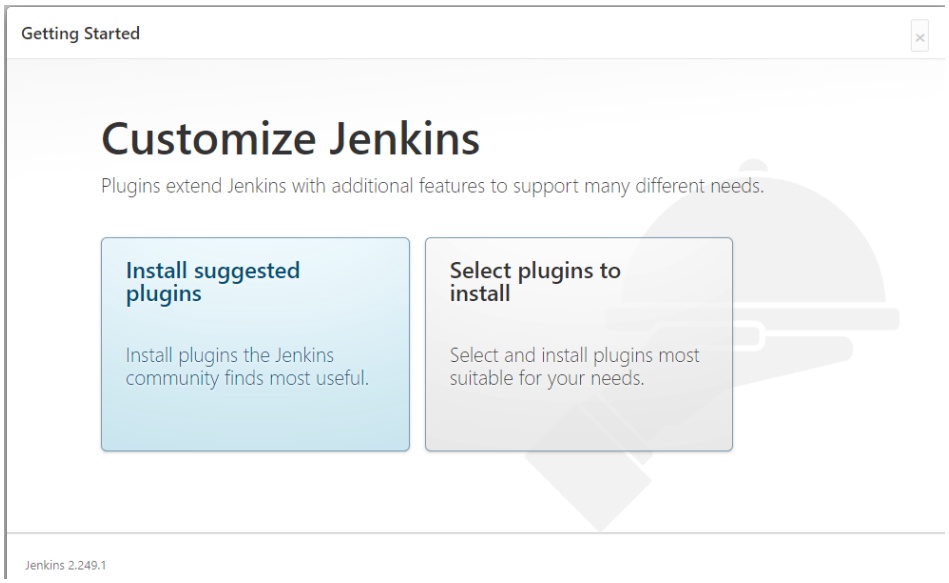
ANEXO 2

Realizar instalação do jenkins https://www.jenkins.io/download/.

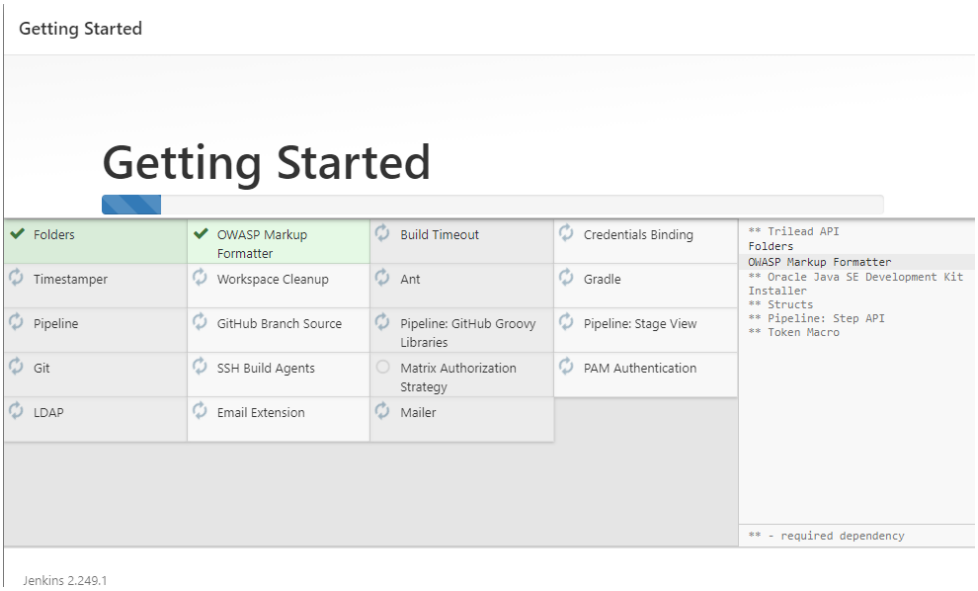
A meio da instalação é necessário desbloquear o *jenkins*, de modo a que este seja configurado com segurança pelo administrador, pode encontrar a *password* gravada no log.



Após validação da *password* já e possível instalar os plugins sugeridos pelo sistema.



ANEXO 2



Após instalação dos plugins iniciais, o *jenkins* está pronto a ser usado.

ANEXO 3

Especificação dos casos de testes desenvolvidos no projeto.

Tipo	Cenário de teste	Condições Necessárias	Descrição do Cenário	Resultado Esperado
Fluxo - Transferências	Transferências com sucesso entre Directo e Directo	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Participantes com saldos disponiveis para realizar a operação - Bancos sem indisponibilidade agendada	O Banco ordenante deverá enviar um pedido de transferência, através do envio de uma mensagem de nota de transferência. Quando a mensagem chegar ao banco beneficiário, este deverá responder com uma resposta de ack positiva. Esta mensagem deverá ser processada e integrada no sistema para que depois possa ser entregue ao Banco ordenante, no final deverá ser enviado um ack de volta para o BB.	Transação que chegou por parte do banco beneficiário fica no estado concluído com sucesso. A transação de aceitação por parte do banco beneficiário foi entregue ao banco ordenante com sucesso. Banco beneficiário recebe o ack enviado pela Sibs. Existe uma reserva antes do envio da nota de transferência para o banco beneficiário Existe movimentação de saldos de uma conta para a outra no final do processo. Verificação do envio da mensagem para a BAM

ANEXO 3

Fluxo - Transferências	Transferências com sucesso entre Directo e o seu Indirecto	- Bancos registados na aplicação IPS - Banco Indirecto com ligação ao banco directo - Ambos participantes têm de estar num estado que permita realizar a operação - Participantes com saldos disponíveis para realizar a operação - Bancos sem indisponibilidade agendada - Indirecto com CREL definido	O Banco ordenante deverá enviar um pedido de transferência, através do envio de uma mensagem de nota de transferência. Quando a mensagem chegar ao banco beneficiário, este deverá responder com uma resposta de ack positiva. Esta mensagem deverá ser processada e integrada no sistema para que depois possa ser entregue ao Banco ordenante, no final deverá ser enviado um ack de volta para o BB.	Transação que chegou por parte do banco beneficiário fica no estado concluído com sucesso. A transação de aceitação por parte do banco beneficiário foi entregue ao banco ordenante com sucesso. Banco beneficiário recebe o ack enviado pela Sibs. Não existe uma reserva antes do envio da nota de transferência para o banco beneficiário Existe movimentação na conta técnica do indirecto e na conta do directo, contudo o salto total agregado do Directo não pode alterar (BALANCE_V) Verificação do envio da mensagem para a BAM
Fluxo - Transferências	Transferências com sucesso entre Indirecto e outro Indirecto	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num	O Banco ordenante deverá enviar um pedido de transferência, através do envio de uma mensagem de nota de transferência. Quando a	Transação que chegou por parte do banco beneficiário fica no estado concluído com sucesso. A transação de aceitação por parte do banco beneficiário foi entregue ao banco ordenante com sucesso. Banco beneficiário recebe o ack enviado pela Sibs. Existe uma reserva antes do envio da nota de transferência para o banco beneficiário

		estado que permita realizar a operação - Participantes com saldos disponiveis para realizar a operação - Bancos sem indisponibilidade agendada - Indirecto com CREL definido	mensagem chegar ao banco beneficiário, este deverá responder com uma resposta de ack positiva. Esta mensagem deverá ser processada e integrada no sistema para que depois possa ser entregue ao Banco ordenante, no final deverá ser enviado um ack de volta para o BB.	Existe movimentação na conta técnica do indirecto e na conta do directo, validar que existe uma redução no saldo do direto associado ao indireto BO, e uma adição do saldo no direto em que está associado o indireto (BALANCE_V) Verificação do envio da mensagem para a BAM
Fluxo - Transferências	Transferências com sucesso entre Indirecto e o seu Directo	- Bancos registados na aplicação IPS - Banco Indirecto com ligação ao banco directo - Ambos participantes têm de estar num estado que permita realizar a operação - Participantes com saldos disponiv	O Banco ordenante deverá enviar um pedido de transferência, através do envio de uma mensagem de nota de transferência. Quando a mensagem chegar ao banco beneficiário, este deverá responder com uma resposta de ack positiva. Esta mensagem deverá ser processada e integrada no sistema para que depois possa ser entregue ao Banco ordenante, no final deverá ser enviado um ack de	Transação que chegou por parte do banco beneficiário fica no estado concluído com sucesso. A transação de aceitação por parte do banco beneficiário foi entregue ao banco ordenante com sucesso. Banco benificiário recebe o ack enviado pela Sibs. Não existe uma reserva antes do envio da nota de transferência para o banco beneficiário Existe movimentação na conta técnica do indirecto e na conta do directo, contudo o salto total agregado do Directo não pode alterar (BALANCE_V) Verificação do envio da mensagem para a BAM

		eis para realizar a operação - Bancos sem indisponibilidade agendada - Indirecto com CREL definido	volta para o BB.	
Fluxo - Transferências	Transferência onde o Banco Beneficiário não responde e a transação caí por Timeout	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Participantes com saldos disponiveis para realizar a operação - Bancos sem indisponibilidade agendada	O Banco ordenante deverá enviar um pedido de transferência, através do envio de uma mensagem de nota de transferência. Quando a mensagem chegar ao banco beneficiário este não responde.	Transação que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Transação de timeout enviada ao banco Beneficiário Transação de rejeição para o banco ordenante Banco benificiário recebe o ack enviado pela Sibs. E uma reserva antes do envio da nota de transferência para o banco beneficiário Validar que deu timeout na CT enviada Existe movimentação na conta BO retirando valor para a reserva, e após timeout o valor deve retornar ao BO (BALANCE_V) Verificação do envio da mensagem para a BAM

Fluxo - Transferências	Transferência onde o Banco Beneficiário responde com uma rejeição	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Participantes com saldos disponiveis para realizar a operação - Bancos sem indisponibilidade agendada	O Banco ordenante deverá enviar um pedido de transferência, através do envio de uma mensagem de nota de transferência. Quando a mensagem chegar ao banco beneficiário este responde com uma rejeição.	Transação que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Transação de rejeição para o banco ordenante Banco benificiário recebe o ack enviado pela Sibs. E uma reserva antes do envio da nota de transferência para o banco beneficiário Existe movimentação na conta BO retirando valor para a reserva, e após timeout o valor deve retornar ao BO (BALANCE_V) Verificação do envio da mensagem para a BAM
Fluxo - Transferências	Transferência onde o Banco Ordenante envia uma transação fora de tempo	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Envio de uma data errada na credit transfer	O Banco ordenante deverá enviar um pedido de transferência, através do envio de uma mensagem de nota de transferência. Após validação por parte da sibs a transação não é enviada para o banco beneficiário uma vez que esta não contem a data correta.	Sibs rejeitou de forma correta a transação. Transação de rejeição para o banco ordenante E uma reserva antes do envio da nota de transferência para o banco beneficiário Existe movimentação na conta BO retirando valor para a reserva, e após timeout o valor deve retornar ao BO (BALANCE_V) Verificação do envio da mensagem para a BAM

ANEXO 3

Fluxo - Recall	Recall sobre uma transferência com sucesso entre Directo e Directo	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de uma transferência entre os bancos realizada com sucesso	O Banco ordenante deverá enviar um pedido de devolução, através do envio de uma mensagem de nota de devolução Após validação por parte da sibs a transação é enviada para o banco beneficiário.	Recall que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Não existe reserva antes do envio da nota de recall para o banco beneficiário Verificação do envio da mensagem para a BAM
Fluxo - Recall	Recall sobre uma transferência com sucesso entre Indirecto e Directo	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de uma transferência entre os bancos realizada com sucesso	O Banco ordenante deverá enviar um pedido de devolução, através do envio de uma mensagem de nota de devolução Após validação por parte da sibs a transação é enviada para o banco beneficiario.	Recall que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Não existe reserva antes do envio da nota de recall para o banco beneficiário Verificação do envio da mensagem para a BAM

Fluxo - Recall	Recall sobre uma transferência com sucesso entre Directo e Indirecto	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de uma transferência entre os bancos realizada com sucesso	O Banco ordenante deverá enviar um pedido de devolução, através do envio de uma mensagem de nota de devolução Após validação por parte da sibs a transação é enviada para o banco beneficiário.	Recall que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Não existe reserva antes do envio da nota de recall para o banco beneficiário Verificação do envio da mensagem para a BAM
Fluxo - Recall	Recall sobre uma transferência com sucesso entre Indirecto e Indirecto	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de uma transferência entre os bancos realizada com sucesso	O Banco ordenante deverá enviar um pedido de devolução, através do envio de uma mensagem de nota de devolução Após validação por parte da sibs a transação é enviada para o banco beneficiario.	Recall que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Não existe reserva antes do envio da nota de recall para o banco beneficiário Verificação do envio da mensagem para a BAM

ANEXO 3

Fluxo - Recall	Recall sobre uma transferência com Id's da transação original errados	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Envio de dados errados na mensagem	O Banco ordenante deverá enviar um pedido de devolução, através do envio de uma mensagem de nota de devolução Após validação por parte da sibs a transação não é enviada para o banco beneficiario, e a sibs envia uma rejeição para o banco ordenante.	Recall que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Banco ordenante vai receber uma resposta negativa a recall que realizou. Não existe reserva antes do envio da nota de recall para o banco beneficiário Verificação do envio da mensagem para a BAM
Fluxo - Respostas Positivas a Recall	Resposta positiva a um pedido de recall com sucesso, aceite pelo FTM	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de um pedido de estorno realizado com sucesso entre os bancos envolvidos - Bancos	Após validação por parte da sibs a transação é enviada para o banco Ordenante Vai haver reserva no BB	Resposta positiva que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM

		com saldos disponiveis para realizar a operação		
Fluxo - Respostas Positivas a Recall	Resposta positiva a um pedido de recall com erro	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de um pedido de estorno realizado com erros - Envio de dados errados (contas/IBAN's)	Após validação por parte da sibs a transação é enviada uma rejeição a resposta da recall ao BB.	Resposta positiva com erro não chega ao BO. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM
Fluxo - Respostas Positivas a Recall	Responder várias 004 para uma recall que já tenha sido respondida com sucesso	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita	Após validação por parte da sibs a transação é enviada uma rejeição a resposta da recall ao BB.	Resposta positiva não chega ao BO. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM

		realizar a operação - Existência de uma resposta positiva a um pedido de estorno pelo ordenante e realizado com sucesso		
Fluxo - Respostas positivas a recall's pelo ordenante	Resposta positiva a um pedido de recall pelo ordenante com sucesso, aceite pelo FTM	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de um pedido de estorno pelo ordenante e realizado com sucesso entre os bancos envolvidos - Bancos com	Após validação por parte da sibs a transação é enviada para o banco Ordenante Vai haver reserva no BB	Resposta positiva que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM

		saldos disponiveis para realizar a operação		
Fluxo - Respostas positivas a recall's pelo ordenante	Resposta positiva a um pedido de recall pelo ordenante com erro	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de um pedido de estorno pelo ordenante e realizado com erros - Envio de dados errados (contas/IBAN's)	Após validação por parte da sibs a transação é enviada uma rejeição a resposta da recall ao BB.	Resposta positiva com erro não chega ao BO. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM
Fluxo - Respostas positivas a recall's pelo ordenante	Responder várias 004 para uma recall pelo ordenante que já tenha sido respondida com	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num	Após validação por parte da sibs a transação é enviada para o banco Ordenante Vai haver reserva no BB	Resposta positiva não chega ao BO. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM

	sucesso	estado que permita realizar a operação - Existência de uma resposta positiva a um pedido de estorno pelo ordenante e realizado com sucesso		
Fluxo - Respostas Negativas a Recall	Resposta negativa a um pedido de recall com sucesso, aceite pelo FTM	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de um pedido de estorno realizado com sucesso entre os bancos envolvidos	Após validação por parte da sibs a transação é enviada a resposta da recall ao BO.	Resposta negativa que chegou por parte do banco beneficiário fica no estado concluído com sucesso. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM

Fluxo - Respostas Negativas a Recall	Resposta negativa a um pedido de recall com erro	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de um pedido de estorno com erro	Após validação por parte da sibs a transação é enviada uma rejeição a resposta da recall ao BB.	Resposta negativa com erro não chega ao BO. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM
Fluxo - Respostas Negativas a Recall	Responder várias 029 para uma recall que já tenha sido respondida com sucesso	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de uma resposta negativa a um pedido de estorno realizado com sucesso	Após validação por parte da sibs a transação é enviada uma rejeição a resposta da recall ao BB.	Resposta negativa não chega ao BO. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM

Fluxo - Respostas negativas a recall's pelo ordenante	Resposta negativa a um pedido de recall pelo ordenante com sucesso, aceite pelo FTM	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de um pedido de estorno pelo ordenante e realizado com sucesso entre os bancos envolvidos	Após validação por parte da sibs a transação é enviada a resposta da recall ao BO.	Resposta positiva não chega ao BO. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM
Fluxo - Respostas negativas a recall's pelo ordenante	Resposta negativa a um pedido de recall pelo ordenante com erro	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de um pedido	Após validação por parte da sibs a transação é enviada uma rejeição a resposta da recall ao BB.	Resposta negativa com erro não chega ao BO. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM

		de estorno pelo ordenante com erro		
Fluxo - Respostas negativas a recall's pelo ordenante	Responder várias 029 para uma recall pelo ordenante que já tenha sido respondida com sucesso	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação - Existência de uma resposta negativa a um pedido de estorno pelo ordenante realizado com sucesso	Após validação por parte da sibs a transação é enviada uma rejeição a resposta da recall ao BB.	Resposta negativa não chega ao BO. Não existe movimentação de saldos de uma conta para a outra no final do processo. (BALANCE_V) Verificação do envio da mensagem para a BAM
Fluxo - Pedido de Investigação	FTM tem a resposta txn e responde com a resposta	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que	O BO envia pedido de investigação	o BB não recebe a mensagem a sibs envia a resposta ao pedido de transferencia para o BO

		permita realizar a operação		
Fluxo - Pedido de Investigação	PDI com info errada e rejeitamos com 002	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação	O BO envia pedido de investigação	o BB não recebe a mensagem a sibs envia a resposta de reição ao pedido de insvestigação para o BO
Fluxo - Pedido de Investigação	FTM não tem a resposta txn e faz forward da resposta	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação	O BO envia pedido de investigação	O BO não reecebe resposta e a sibs vai enviar o pedido de investigação para o BB
Fluxo - Pedido de Status	FTM tem a resposta de recall e responde com a resposta	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que	O BO envia pedido de status	o BB não recebe a mensagem a sibs envia a resposta ao pedido de statuscom a resposta da recall para o BO

		permita realizar a operação		
Fluxo - Pedido de Status	PDI com info errada e rejeitamos com 029	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação	O BO envia pedido de status	o BB não recebe a mensagem a sibs vai validar e rejeitar o pedido de status com uma 029
Fluxo - Pedido de Status	FTM não tem a resposta txn e faz forward da resposta para o BB	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação	O BO envia pedido de status	O BO não reecebe resposta e a sibs vai enviar o pedido de status para o BB
Fluxo - Pedido de Status pelo Ordenante	FTM tem a resposta da recall pelo ordenante e responde com a resposta	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que	O BO envia pedido de status	o BB não recebe a mensagem a sibs envia a resposta ao pedido de status com a resposta da recall para o BO

		permita realizar a operação		
Fluxo - Pedido de Status pelo Ordenante	PDI com info errada e rejeitamos com 029	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação	O BO envia pedido de status	o BB não recebe a mensagem a sibs vai validar e rejeitar o pedido de status com uma 029
Fluxo - Pedido de Status pelo Ordenante	FTM não tem a resposta txn e faz forward da resposta para o BB	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação	O BO envia pedido de status	O BO não reecebe resposta e a sibs vai enviar o pedido de status para o BB
Sistema - Validar servidor tem espaço livre			Efectuar login via ssh para o servidor IIB. Efetuar o seguinte comando: df -F %z /var/mqsi/	Percentagem usada deve ser inferior a 100%. Exemplo de output: Filesystem 512-blocks %Used /dev/fslv03 14680064 29%

Sistema - Validar nó do Bus	Validar se o nó esta a correr (mqsilist | mqsilist %NODE% -e %EXEC_GRP%)		Efetuar login via ssh para servidor IIB. Executar o seguinte comando: mqsilist NODEFTMDEV21 Sendo que "NODEFTMDEV21 " é o nome do Message Broker em DEV	Exemplo: "BIP1286I: Execution group 'FTM' on broker 'NODEFTMDEV21' is running." Haverá o número de linhas igual ao número de execution groups, tendo o estado que estar em "running".
Sistema - Validar Remoção de CoreDumps	/var/mqsi | sftp://172.23.97.133 | mqm | 1qaz2wsx | porto 22	Devem existir resultados na execução do teste anterior (Sistema - Validar Geração de CoreDumps)	Efetuar login via ssh para servidor IIB. Executar os seguintes comandos: rm -f /var/mqsi/core* 2> /dev/null rm -f /var/mqsi/common/errors/core* 2> /dev/null De seguida executar os seguintes: ls -Al /var/mqsi/core* 2> /dev/null | tr -s ' ' | cut -f9- -d' ' ls -Al /var/mqsi/common/errors/core* 2> /dev/null | tr -s ' ' | cut -f9- -d' '	Não deve ser retornado output
Sistema - Validar Geração de CoreDumps	/var/mqsi | sftp://172.23.97.133 | mqm | 1qaz2wsx | porto 22		Efetuar login via ssh para servidor IIB. Executar os seguintes comandos: ls -Al /var/mqsi/core* 2> /dev/null | tr -s ' ' | cut -f9- -d' ' ls -Al /var/mqsi/common/errors/core* 2>	Não deve ser retornado output

			/dev/null | tr -s ' ' | cut -f9- -d' '	
Portal	Realização de login no portal com banco SIBS	- Bancos registados na aplicação IPS - Utilização do banco SIBS	Para efetuar login no portal deve ser criado o token com os seguintes valores: COM_NUM = 1 EXT_BIC = SISRPTP0XXX - Validar na BD que o BIC utilizado é do tipo SIBS - Validar que o login foi efetuado com sucesso no portal IPS.	O banco é do tipo SIBS - O login foi efetuado com sucesso no portal IPS
Portal	Realização de login no portal como banco	- Bancos registados na aplicação IPS - Utilização do bic de um banco existente que não seja SIBS	Para efetuar login no portal deve ser criado o token com os seguintes valores: COM_NUM = 0 EXT_BIC = BANKTESTECA - Validar na BD que o BIC utilizado não é do tipo SIBS - Validar que o login foi efetuado com sucesso no portal IPS.	O banconao é do tipo SIBS - O login foi efetuado com sucesso no portal IPS
Portal	Criar indisponibilidade como SIBS	Bancos registados na aplicação IPS -Ambos participantes têm de estar num estado que	Para efetuar login no portal deve ser criado o token com os seguintes valores: COM_NUM = 1 EXT_BIC = SISRPTP0XXX - Validar na BD que o BIC utilizado é do tipo SIBS - Selecionar o	Banco é do tipo SIBS. É possivel criar uma insdisponibilidade com sucesso.

		permita realizar a operação	Menu: Dados Estáticos > Calendário do IPS > Indisponibilidade do Participante - Pesquisar BIC do banco onde quer adicionar a indisponibilidade - Cliar em Adicionar, e configurar a hora de abertura e fecho e os dias da semana. - Validar que foi criada a indisponibilidade com sucesso.	
Portal	Eliminar indisponibilidade como SIBS	Bancos registados na aplicação IPS -Ambos participantes têm de estar num estado que permita realizar a operação	Para efetuar login no portal deve ser criado o token com os seguintes valores: COM_NUM = 1 EXT_BIC = SISRPTP0XXX - Validar na BD que o BIC utilizado é do tipo SIBS - Selecionar o Menu: Dados Estáticos > Calendário do IPS > Indisponibilidade do Participante - Pesquisar BIC do banco onde quer eliminar a indisponibilidade - Validar a linha da tabela que pretende eliminar - Clicar no icon do lixo e clicar no botão SIM. - Validar que a indisponibilidade	Banco é do tipo SIBS. É possivel Eliminar uma insdisponibilidade com sucesso.

			foi removida com sucesso.	
Portal	Eliminar indisponibilidade como Banco	Bancos registados na aplicação IPS -Ambos participantes têm de estar num estado que permita realizar a operação	Para efetuar login no portal deve ser criado o token com os seguintes valores: COM_NUM =0 EXT_BIC = BANKTESTECA - Validar na BD que o BIC utilizado é do tipo Banco - Selecionar o Menu: Dados Estáticos > Calendário do IPS > Indisponibilidade do Participante - Validar a linha da tabela que pretende eliminar - Clicar no icon do lixo e clicar no botão SIM. - Validar que a indisponibilidade foi removida com sucesso.	Banco é do tipo Banco É possivel eliminar uma insdisponibilidade com sucesso.
Portal	Criar indisponibilidade como Banco	Bancos registados na aplicação IPS -Ambos participantes têm de estar num estado que permita realizar a	Para efetuar login no portal deve ser criado o token com os seguintes valores: COM_NUM = 0 EXT_BIC = BANKTESTECA - Validar na BD que o BIC utilizado é do tipo Banco - Selecionar o Menu: Dados Estáticos >	Banco é do tipo Banco É possivel criar uma insdisponibilidade com sucesso.

		operação	Calendário do IPS > Indisponibilidade do Participante - Cliar em Adicionar, e configurar a hora de abertura e fecho e os dias da semana. - Validar que foi criada a indisponibilidade com sucesso.	
Portal	Navegar pelos menus, login realizado como SIBS	- Bancos registados na aplicação IPS - Utilização do banco SIBS	Para efetuar login no portal deve ser criado o token com os seguintes valores: COM_NUM = 1 EXT_BIC = SISRPTP0XXX - Validar na BD que o BIC utilizado é do tipo SIBS - Validar Menu Conta registo - Transações > Pesquisa - Conta > Saldo de conta - Conta > Movimentos da Conta - Conta > Reconciliação - Limites > Limites de Transações - Validar Menu Serviços - Gestão de Risco > Limite de Crédito - Dados de Negócio > Gráficos - Dados de Negócio > Os	O login foi efetuado com sucesso no portal IPS Os menus descritor no cenários estão disponiveis no portal.

			meus relatorios - Validar Menu Dados estaticos - Participantes > Pesquisa - Participantes > Gerir - Participantes > Tabela de Roteamento - Gerir CSM > Gerir CSM - Gerir CSM > SIBS - Calendaio IPS > Calendário Target2 - Calendário IPS > Indisponibilidade do participante - Validar Menu Administração - Segurança > Auditar Registos - Segurança > Gestão de perfis - Alertas > Pesquisa/Gerir - Alertas > Mensagens - Dados Históricos > Obter Dados Históricos - Configuração > Eventos	

Portal	Navegar pelos menus, login realizado como Banco	- Bancos registados na aplicação IPS - Utilização do bic de um banco existente que não seja SIBS	Para efetuar login no portal deve ser criado o token com os seguintes valores: COM_NUM = 1 EXT_BIC = SISRPTP0XXX - Validar na BD que o BIC utilizado não é do tipo SIBS - Dentro do menu Conta Registo > Transações > Pesquisa validar que não existe o BIC a pesquisar. - Dentro do menu Conta Registo > Conta > Saldo de conta validar que não existe o BIC a pesquisar. - Dentro do menu Conta Registo > Conta > Movimentos da Conta validar que não existe o Bic a pesquisar. - Validar que não existe o menu Conta Registo > Conta > Reconciliação. - Dentro do menu Serviços > Dados de Negócio > Gráficos validar que a Tab bic CSM nao está presente na pagina. - Validar que existe o menu Serviços > Dados de Negócio > Os meus relatorios. - Dentro do menu Dados Estáticos Participantes Pesquisa validar	O login foi efetuado com sucesso no portal IPS Os menus descritores nos cenários estão disponíveis/indisponíveis no portal.

			que não existe a possibilidade de pesquisa. - Validar que nao existe o sub menu Gerir (Dados estaticos > Participantes > Gerir) - Dentro do menu Dados estaticos > Participantes > Tabela de Roteamento validar que a tab tabela de roteamento não está presente na página. - Dentro do ,emu Dados estáticos validar que nao existe o sub menu Gerir CSM. - Validar que existe o menu Administracao > Segurança > Auditar Registos - Dentro do menu Administracao > Segurança validar que nao existe o sub menu Gestão de perfis. - Dentro do Menu Administracao > Alertas > Pesquisa/Gerir validar que não é possivel realizar uma pesquisa. - Validar que existe o menu Administracao > Alertas > Mensagens. - Dentro do menu Administracao > Dados Históricos > Obter Dados	

			Históricos validar que não existe a tab de reconciliação. - Validar que existe o menu Administracao > Configuração > Eventos.	
Portal	Pesquisa de Transacções	-Bancos registados na aplicação IPS. - Ambos participantes têm de estar num estado que permita realizar a operação. - Participantes com saldos disponivel. - Utilização do bic de um banco existente que não seja SIBS. - Realização de transacções para o banco especifico	Usar aplicação powershell para realizar algumas transacções. - Selecionar o Menu: Conta Registo > Transacções > Pesquisa Especificar filtros de pesquisa que vão de encontro com as transacções realizadas anteriormente. Verificar lista de resultados para encontrar um ID conhecido. Entrar nos detalhes. Comparar valores.	Dados dos detalhes batem certo com dados em base de dados.

ANEXO 3

Portal	Pesquisa de Movimentos	-Bancos registados na aplicação IPS. - Ambos participantes têm de estar num estado que permita realizar a operação. - Participantes com saldos disponivel. - Utilização do bic de um banco existente que não seja SIBS. - Realização de transacções para o banco especifico	Usar aplicação powershell para realizar algumas transacções. - Selecionar o Menu: Conta Registo > Conta > Movimentos da Conta Especificar filtros de pesquisa que vão de encontro com as transacções realizadas anteriormente. Verificar lista de resultados para encontrar um ID conhecido. Entrar nos detalhes. Comparar valores.	Dados dos detalhes batem certo com dados em base de dados.
Portal	Dashboards	'- Existência de transacções realizadas com sucesso para um banco.	- Selecionar o Menu: Serviços > Dados de Negócio > Gráficos Selecionar um banco e um período de tempo onde o banco tenha transações com sucesso.	10 dashboards com informação. 5 Valores / 5 Números. Valor - Montante / Dinheiro Número - Nº de transacções

Portal	Relatórios	'- Existência de transacções realizadas com sucesso para um banco.	Usar aplicação powershell para realizar algumas transações. - Selecionar o Menu: Conta Registo > Transacções > Pesquisa Especificar filtros de pesquisa que vão de encontro com as transacções realizadas anteriormente. Verificar se existe resultados na lista. Clicar Export > PDF. Após sucesso ir ao Menu: Serviços > Dados de Negócio > Os meus relatórios. Fazer download do relatório em questão.	PDF com informação.
Ficheiros - YEXT	Geração de um ficheiro diário	- Bancos registados na aplicação IPS - Ambos participantes têm de estar num estado que permita realizar a operação	Entrar nos detalhes.	Ficheiro com vários registos. Necessário validar registos com base de dados usando o excel "Ficheiros_YEXT V06 CR2" para fazer o mapeamento entre BD e Ficheiro.

Quadro 2 - Especificação dos casos de teste