UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Quality Assurance

Maria de Fátima Abrunhosa Fernandes

Mestrado em Engenharia Informática

Trabalho de Projeto orientada por:
Wellington de Oliveira Júnior

2022

Agradecimentos

Começo por agradecer ao Professor Wellington de Oliveira Júnior, não só pela sua orientação ao longo do meu projeto, mas também pela sua disponibilização, rapidez e paciência a responder todas as minhas dúvidas sobre as partes burocráticas da minha tese, mesmo nos primeiros dias após ter chegado a Portugal.

No âmbito do estágio realizado durante o projeto, gostaria de agradecer ao meu supervisor João Catalão, por estar sempre disponível para qualquer questão que eu tive sobre o trabalho que me foi encarregue de elaborar, mas também por não ser leve demais com a carga, por me ter conseguido trazer ao de cima as minhas habilidades como *tester*.

Quero agradecer ao Miguel Simões e ao Sérgio Sequeira por me ensinarem os básicos do trabalho que desenvolvi e por acreditarem na minha ética de trabalho sempre que me encarregavam de um projeto.

Também tenho de agradecer ao Pedro Fernandes e ao Carlos Serra, que me explicaram as funcionalidades dos módulos que tive de testar automaticamente e esclareceram todas as minhas dúvidas com toda a clareza, mesmo as mais óbvias, e ao Paulo Belo que teve de validar os meus documentos de 60-90 páginas.

Um grande agradecimento à Leonor Tique e ao Filipe Grangeiro por me darem a oportunidade de realizar a minha tese na *ARTSOFT*.

Tenho de agradecer a todos os colaboradores da *ARTSOFT* por me darem as boas-vindas de abraços abertos e me fazerem sentir parte da equipa.

À minha melhor amiga Daniela, que sempre ouviu os meus desabafos e demonstrou apoio na realização dos meus objetivos, mesmo quando eu própria não conseguia ver as conquistas que estava a realizar ao longo do meu trabalho.

Quero também deixar os meus agradecimentos a minha vizinha Adélia, que sempre me tratou como se eu fosse mais uma filha dela, dando-me sempre ajuda quando eu mais precisava e ajudando a tratar da casa quando eu demasiado ocupada para tratar dos detalhes.

À minha psicóloga Sofia, que sem as suas conversas e incentivos não teria tido força para acabar o mestrado e não teria tido a felicidade de chegar a este ponto.

E por fim, quero deixar um enorme agradecimento ao meu pai, que sempre celebrou as minhas conquistas, sempre me disse que eu era capaz, sempre acreditou em mim e fez de tudo para que eu acreditasse também, sempre que me sentia mal, disse-me que não era o fim do mundo, que tudo se ia compor porque eu era capaz e sempre se certificou que eu tinha tudo ao meu dispor para poder trabalhar na minha tese sem ter de pensar em coisas extra.

Resumo

O *software* do *ARTSOFT* é um *ERP* (*enterprise resource planning*), em português planeamento dos recursos empresariais, um programa que ajuda empresas a gerir as suas atividades relacionadas com: contabilidade; recursos humanos; projetos; produção; entre outras.

Neste *software*, esteve-se encarregue de testar os módulos, secções do programa: *workflows*; produção e assistências técnicas.

Um *workflow* é uma máquina de estados que armazena o estado de um objeto em um determinado momento e pode mudá-lo através de ações baseadas no *input* que recebe. No *ARTSOFT*, os *workflows* são usados para gerir fluxos de trabalho de uma empresa com o objetivo de automatizar os processos desta e aumentar a eficiência de trabalho.

O módulo de produção gere a estruturação dos processos de produção de materiais, que consiste no planeamento, organização e controlo dos *stocks* dos diversos materiais tal como a coordenação dos pedidos dos clientes dos produtos finais. Desta forma, uma empresa pode controlar efetivamente quanto *stock* é que precisam de um determinado material e quando o precisam de começar a produzir para conseguirem entregar ao cliente o produto. O módulo também permite definir tarefas para cada trabalhador, definindo o seu papel na produção de uma ordem, e registar quaisquer custos relacionados com a produção dos materiais.

As assistências técnicas, referem-se a serviços pós-venda, como reparações; substituições; revisões; entre outros. Assim, no *ARTSOFT*, foi elaborado este módulo para ajudar na gestão dos equipamentos sobre os quais os serviços vão ser realizados os serviços; na gestão das informações dos clientes associados aos equipamentos; na gestão do estado em que o serviço está e os custos que vão sendo acumulados devido ao serviço.

Quality Assurance é um processo que garante aos clientes um *software* com o nível de qualidade pretendido, ou seja, tem os requisitos pedidos pelos clientes e os mesmos não tenham quaisquer erros e tenham um funcionamento correto. Assim, o trabalho de um *tester* é desenvolver os testes necessários, automáticos ou manuais e *Black-Box* ou *White-box*, conforme o que seja melhor para o *software* em questão. De forma a poder escolher qual a melhor forma de testar um programa ou uma certa funcionalidade é preciso saber quais as vantagens e desvantagens entre os testes manuais e os testes automáticos. Contudo, é normal ser usado uma combinação entre os dois tipos de teste.

Em *black-box* é possível encontrar *bugs* que os programadores nem sequer tinham pensado em testar, como o programa está a ser testado como um utilizador o utilizaria. Também é possível encontrar erros enquanto o *software* está em desenvolvimento, ajudando a poupar tempo aos programadores, visto que podem continuar o desenvolvimento com os erros identificados em mente. Contudo, muitas das vezes pode-se acabar por gastar mais tempo do que o suposto pois esteve-se a testar um comportamento que não era um requisito da aplicação. Para evitar esta situação, convém a equipa de programadores e *testers* estarem continuamente a partilhar conhecimento entre si.

Para se poder testar a qualidade dos módulos foram elaborados ficheiros de *scripts*, código de linguagem *C* que contêm o automatismo para testar todas as funcionalidades existentes nos módulos.

Os testes automáticos são uma forma de teste em que são usadas ferramentas para executar os casos de teste, sem ser preciso interferência humana. Com estas ferramentas podemos abordar diversos *inputs* permitidos, e não permitidos, pelo programa a testar e validar o resultado obtido com o esperado. Logo, permitem a testagem de casos de teste repetitivos e com ligação a vários serviços, que, caso contrário, levariam bastante mais tempo a testar.

Para além destes testes automáticos, realizou-se testes manuais de *black-box* sempre que era feita uma correção de um bug reportado pelos parceiros/clientes.

Ao longo do projeto, foram atribuídos eventos diariamente para proceder à testagem das alterações feitas. A testagem foi elaborada através de testes manuais, onde é necessário ter conhecimentos profundos sobre a funcionalidade sobre a qual se está a testar. Desta forma, se quando começa a analisar

um evento, para além de se analisar o contexto do mesmo, é necessário ir à documentação do programa para estudar qual o comportamento esperado do *software* tal como que outras funcionalidades podem ter sido afetadas com as alterações efetuadas. À medida que se vai analisando a documentação, num ficheiro à parte, anota-se os casos de teste a realizar.
Após ter os casos de teste descritos, é preciso ter o ambiente de teste preparado para os executar. Isto envolve preparar uma base de dados, com as configurações e os critérios que são indicados no evento. Neste projeto, os meus ambientes de testes consistiram em cópias das bases de dados dos clientes, para ter a certeza de que estou a simular os passos que os futuros utilizadores iram seguir.
Com os casos de teste e o ambiente de teste preparados, executam-se os testes. No caso de eventos, os testes elaborados foram do tipo *black-box*, que, como descrito na Subsecção 2.2.1, são testes realizados do ponto de vista do cliente sem o *tester* ter acesso à parte interna do *software*. No fim da testagem, registam-se os resultados obtidos e comparam-se com os esperados para verificar que o programa está a funcionar corretamente.
Quando dos testes se obtêm resultados corretos é dado como concluído a tarefa de testagem e transfere-se o evento para que as alterações possam ser disponibilizadas aos clientes nas próximas versões do programa. Por outro lado, se os resultados esperados forem diferentes dos obtidos é preciso rever a suposta correção ou alteração feita ao código. Para isso o evento é transferido para o programador que realizou as modificações para que possam verificar a situação e corrigir.
Como o ARTSOFT é um software em que os vários módulos estão interligados entre si com bastantes configurações, antes de se poder realizar testes, teve-se de ter uma formação sobre as funcionalidades do programa. O mesmo ocorreu antes de começar a elaborar ficheiros de *scripts*. Houve uma formação sobre o *software* de testes que é usado pela empresa, *TestComplete*. O *TestComplete* é um programa de testes automatizados de UI (user interface), em português interface do utilizador, desenvolvido pela Smartbear. Neste *software* foram desenvolvidos os *scripts* e corrigidos sobre todas as versões do programa *ARTSOFT* que estivessem para sair, verificando se ocorria algum erro na sua execução.
Com o desenvolvimento de técnicas de testagem de *software*, várias ferramentas capazes de testar diferentes tipos de programas foram realizadas com o objetivo de ajudar os *testers* de *software* na automatização do seu trabalho. No entanto, com várias ferramentas veem múltiplas escolhas de qual utilizar e quando. Assim, nos últimos anos, foram realizados estudos que comparam ferramentas de automatização de testes disponíveis no mercado a escolha de qual usar seja eficaz, mais rápida e certa. Nas próximas Subsecções são descritas três das mais populares ferramentas de executar testes automáticos disponíveis, as suas caraterísticas e, no final, qual a melhor a ser usada no meu projeto e porquê.
O principal objetivo deste projeto, é que todas as versões que sejam entregues aos parceiros e clientes da empresa ARTSOFT não tenham nenhuns bugs, erros no programa, principalmente os módulos a que este projeto se dedica. De modo, a que a qualidade do *software* seja melhorada tal como a impressão dos utilizadores sobre a empresa e o programa. De onde vem a pergunta de pesquisa deste projeto: Os testes e *scripts* aumentam a qualidade do código?

Palavras-chave: *ERP*; *Scripts*; *Workflows*; Produção e Assistências Técnicas.

Abstract

ARTSOFT's software is an ERP (enterprise resource planning), a program that helps the management of a company's activities related to accounting; human resources; projects; productions; among others. A workflow is a state machine that stores the state of an object, at a given moment, and can change it through actions that are based on the input it receives.

The production module manages the structuring of the process of material production, which consists of planning, organizing, and controlling the stocks of various materials, as well as, coordinating the customers' orders of the final products.

Technical assistance refers to after-sales services, such as: repairs, replacements, overhauls, among others. Thus, in ARTSOFT, this module was developed to help in the management of the equipment on which the services will be performed.

To be able to test the quality of these modules I elaborated script files, C language code that contain the automatism to test all the existing functionalities in the modules. In addition to these automated tests, I performed manual black-box tests each time a bug reported by partners/customers was fixed.

I was trained about the testing software that is used by the company, TestComplete. TestComplete is an automated UI (user interface) testing program, developed by Smartbear. In this software I wrote the scripts and ran them over all versions of ARTSOFT that were about to be released, verifying if any error occurred in their execution.

My main task in this project is that all versions are delivered to the partners and customers of the ARTSOFT company without any bugs, errors in the program, mainly the modules that were under my responsibility. So that the quality of the software is improved as well as the users' impression of the company and its software.

Keywords: ERP; Scripts; Workflows; Production and Technical Assistance

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

ERP – *enterprise resource planning*
IU – *interface* do utilizador
BPM - *business process management*
MRP II - *manufacturing resource planning*
MES - *manufacturing execution system*
QA – *quality assurance*
SAFT - *standard audit file for tax purposes*
SS – segurança social
MPP – *materials requirement planning*
XML - *extensible markup language*

1 Introdução

1.1 Motivação

Na criação de um software estão envolvidos, normalmente, mais que um par de mãos humanas, principalmente se o programa a desenvolver for de grande dimensão. Desta forma, nem sempre é fácil de todos os elementos da equipa entenderem os requisitos necessários para a aplicação e de, simplesmente, haver erros de causa humana, e.g., desenvolver uma funcionalidade na versão errada do programa. A situação complica-se quando se juntam os bugs e defeitos da aplicação, que infelizmente vão sendo criados ao mesmo tempo que o código dela. Para manter o software a um nível de alta qualidade é preciso ter alguém que verifique o trabalho da equipa de programadores, alguém que consiga identificar e proteger os clientes dos erros, os *testers* de *Quality Assurance*.

Desta motivação segue a seguinte pergunta de pesquisa: Os testes e *scripts* aumentam a qualidade do código?

1.2 Enquadramento

Como já referido acima *Quality Assurance* é um processo que garante aos clientes um *software* com o nível de qualidade pretendido, ou seja, tem os requisitos pedidos pelos clientes e os mesmos não tenham quaisquer erros e tenham um funcionamento correto. Assim, o trabalho de um *tester* é desenvolver os testes necessários, automáticos ou manuais e *Black-Box* ou *White-box*, conforme o que seja melhor para o *software* em questão.

Neste projeto, o programa que foi objeto de estudo foi o *ARTSOFT ERP*, um programa que ajuda empresas, desde pequenas a grandes, a conseguirem gerir os seus processos de forma mais eficaz. Para tal, fornece os seguintes serviços:

- Gestão comercial;
- Contabilidade e Gestão Financeira;
- Recursos humanos;
- **Gestão de Assiduidades;**
- *Software* de Gestão de Projetos;
- Gestão de Processos (*BPM*);
- Organização Documental;
- **Gestão de Assistências Técnicas;**
- **Gestão de Produção (*MRP*);**
- Gestão de Resíduos.

Estes serviços permitem a uma empresa uma *performance*e eficácia, mais rigorosa nos processos empresariais, uma monitorização constante do que se passa, automatização e sistematização da gestão de informação que é transferida entre vários departamentos.

Entre os serviços acima descritos, este projeto trabalhou detalhadamente com *workflows*, que agilizam o fluxo de processos da empresa, como: i) gestão de assiduidades, ii) gestão de assistências técnicas e iii) gestão de produção. No i) gere-se as férias, presenças, trabalhos suplementares, etc. dos empregados da empresa. O ii) dá apoio às empresas com serviços de pós-venda, como gerir reparações, equipamentos, ordens de serviço e recursos. Por fim, o iii) apresenta uma solução para as necessidades da produção do setor industrial. O serviço de produção abrange *MRP II (manufacturing resource planning*) , em português planeamento de recursos de produção, um conjunto de soluções integradas e

funções relativas ao controle de chão de fábrica e gestão de recursos, e *MES (manufacturing execution system)*, em português sistemas de execução de produção, uma solução que gera informações precisas e em tempo real para otimizar as etapas da produção, desde a criação de uma ordem até o embarque dos produtos acabados (ARTSOFT, 2022).

Para além dos módulos que são o foco deste projeto, foram realizados testes manuais sobre alguns dos outros, como gestão comercial e contabilidade e gestão financeira, para verificar se os erros reportados por clientes foram corrigidos. Nestes testes, não foi preciso ter conhecimento profundo sobre os módulos, ao contrário dos mencionados acima, assim, foi possível testá-los sem que o projeto ficasse para trás e ajudar a empresa a manter a qualidade que os clientes estão habituados.

Este projeto foi desenvolvido ao longo de nove meses, como pode ser verificado pela Tabela 1.1, onde cada tarefa está dividida por mês, sendo M1 o primeiro mês, M2 o segundo e assim por diante. Nos primeiros dois meses, foi dada formação sobre o funcionamento da empresa, sobre técnicas de testagem e sobre o *software* em si. De seguida, foram desenvolvidos os documentos de plano de teste, onde se registou tudo o que teria de ser testado nos serviços, e escreveu-se os *scripts* associados. Depois foi-se dando manutenção aos *scripts* elaborados sempre que havia alguma alteração no código ou uma funcionalidade nova. Por fim, o último mês foi dedicado à escrita deste documento sobre o projeto realizado.

Tabela 1.1 - Cronograma das atividades ao longo do projeto

Atividades	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**
Formação teórica e prática sobre conceitos gerais	X	X							
Estudo e análise da solução a ser implementada			X	X	X				
Manutenção do trabalho a ser desenvolvido						X	X	X	
Elaboração do relatório de fim do projeto									X

1.3 Objetivos

Os serviços que o projeto descreve são de desenvolvimento recente, por parte da equipa de programação da *ARTSOFT*. Isto significa que, as funcionalidades e os requerimentos ainda não foram testados a fundo aquando do começo da tese. Logo, de modo que se consiga serviços com qualidade para entregar aos clientes, foi necessário desenvolver testes de base. Os testes foram efetuados dentro da equipa de *Quality Assurance*, onde foi desenvolvido o projeto.

Devido a serem módulos recentes, a solução teve de passar por todos os passos de testagem, como elaborar documentos de plano de testes, traduzir os casos de teste para *scripts*, código que corre testes automáticos, e até mesmo testes manuais em casos específicos. Tudo isto com o objetivo de que chegue aos clientes um *software* de confiança, funcional e sem *bugs* ou erros graves.

A metodologia vai ser demonstrada com mais pormenor na secção 3.

1.4 Estrutura do documento

A estrutura do documento segue os passos que foram seguidos durante o desenvolvimento do projeto na equipa de *Quality Assurance* da *ARTSOFT*, desde a formação inicial até à implementação de *scripts* sobre três diferentes módulos. Em seguida resume-se o conteúdo de cada capítulo.

Na próxima Secção são expostos os conceitos teóricos inerentes ao *ARTSOFT* e, consequentemente, ao trabalho desenvolvido, nomeadamente a importância de testes manuais e automáticos (Secção 2.1), a diferença entre *Black-Box* e *White-box* (Secção 2.2), o que se entende por *scripts* (Secção 2.3) e a sua utilização de ferramentas de automação como o *TestComplete* (Secção 2.4), a descrição de eventos e como se integram no meu trabalho (Secção 2.5), o que são as bases de dados no contexto do projeto (Secção 2.6), e, por fim, o que significam os conceitos de *workflows* (Secção 2.7.1), assistências técnicas (Secção 2.7.3) e produção (Secção 1.1.1), os módulos principais do projeto.

Na Secção 3 são expostas as técnicas usadas para a concretização da solução deste projeto, como o processo de testagem automática, que é dividida entre dados (Secção 3.1), eventos (Secção 3.3), planos de testes (Secção 3.4) e *scripts* (Secção 3.4).

Na Secção 4 são apresentados os resultados obtidos dos testes que realizei, em especial através da visualização de números concretos de quantos eventos foram desenvolvidos, reportados, transferidos de volta para programador e corrigidos.

Na Secção 5 é apresentada a discussão dos resultados obtidos com as técnicas utilizadas em mente e como as mesmas podem evoluir no futuro.

Na Secção 6 são descritos outros trabalhos sobre testagem automática, em que tenham utilizado ferramentas para realizar os testes. Desta forma é feita uma comparação entre o *software* usado noutros trabalhos e o usado neste projeto, *TestComplete*, com vista a concluir qual a melhor ferramenta de testagem automática no contexto da minha tese.

Por fim, na Secção 7 são resumidas as principais conclusões alcançadas através do projeto, as ameaças que o trabalho pode sofrer (Secção 7.1) e os possíveis trabalhos futuros (Secção 7.2).

2 Conceitos

2.1 Testes manuais *vs.* Testes automáticos

Os testes de *software* é uma das últimas etapas antes de o programa ser entregue ao cliente, mas é uma de extrema importância, visto que é a partir dela que se confirma se os requisitos foram cumpridos e o comportamento é correto. De forma a garantir que todos os requisitos do *software* são testados, são efetuados documentos com os casos de teste a realizar. Existem duas formas principais de realizar os casos de teste, através de testes manuais ou automáticos.

2.1.1 Testes manuais

Os testes manuais são uma forma de testar em que os casos de teste são executados sem o uso de uma ferramenta de automação, ou seja, manualmente. Este tipo de testes é realizado quando uma nova funcionalidade é lançada para os clientes, visto que, normalmente, com ela vem instabilidade e erros.

Como queremos que o cliente tenha uma experiência sem *bugs* e nem tudo é possível testar automaticamente, é necessário, nesses casos, testar o programa através dos olhos do utilizador, manualmente (javatpoint, 2022).

A realização destes testes segue um processo, relativamente, simples, contudo que envolve, bastante estudo do programa. Assim, este processo de testagem é realizado da seguinte forma:

- O *tester* tem de estudar todos os documentos do programa que estejam relacionados com as funcionalidades a testar;
- O *tester* analisa o comportamento que o *ARTSOFT* é suposto ter;
- São desenvolvidos os casos de teste de acordo com o comportamento delineado no passo anterior;
- Os casos de teste são executados;
- Se ocorrerem erros, descreve-se o que ocorreu, em que versões e informa-se a equipa de programadores encarregues para poderem voltar a resolver;
- Quando não houver nenhum erro, descreve-se os resultados obtidos e indica-se que a funcionalidade testada está pronta para ser enviada para os clientes.

2.1.2 Testes automáticos

Os testes automáticos são uma forma de teste em que são usadas ferramentas para executar os casos de teste, sem ser preciso interferência humana. Com estas ferramentas podemos abordar diversos *inputs* permitidos, e não permitidos, pelo programa a testar e validar o resultado obtido com o esperado. Logo, permitem a testagem de casos de teste repetitivos e com ligação a vários serviços, que, caso contrário, levariam bastante mais tempo a testar.

No entanto, apesar de ser um processo em que se poupa tempo a fazer testes exploratórios, testes onde se investiga o programa para identificar erros nas funcionalidades que podem afetar os resultados do sistema, é preciso alocar mais tempo e recursos à manutenção dos casos de teste e os *scripts*.

No caso deste tipo de testes, antes de se começar o processo de testagem, é necessário escolher a ferramenta de automatização apropriada para o *software*. Cada ferramenta tem propriedades únicas que as fazem com que funcionem para testar um programa, mas não o outro. Assim, é preciso estudar as ferramentas disponíveis para ver qual a melhor correspondência (javatpoint, 2022).

Depois de se ter uma ferramenta escolhida, realiza-se o verdadeiro processo de testagem:

- O *tester* tem de estudar todos os documentos do programa que estejam relacionados com as funcionalidades a testar;
- Realizar um documento de plano de teste, onde se elabora todos os casos de teste relativos ao programa a testar;
- Validar o plano de teste com a equipa encarregue da funcionalidade a testar;
- Traduzir o que está escrito no plano de testes em código para se escrever os *scripts*;
- Executar os *scripts* através da ferramenta de automação escolhida;
- Se a ferramenta indicar que houve um erro durante a execução, tem de se investigar a causa e reportar aos programadores para poderem corrigir;
- Quando a ferramenta não der nenhum erro durante a execução, informa-se o resto da equipa que a funcionalidade está pronta para o cliente.

2.1.3 Vantagens e desvantagens

De forma a poder escolher qual a melhor forma de testar um programa ou uma certa funcionalidade é preciso saber quais as vantagens e desvantagens entre os testes manuais e os testes automáticos. Contudo, é normal ser usado uma combinação entre os dois tipos de teste. Por exemplo, o projeto, quando era preciso testar uma pequena funcionalidade que só tinha um ou dois casos de teste, realizava-se testes manuais, pois não valia a apena estar a elaborar um plano de testes e *scripts* para tão poucos casos de teste que, na maior parte do tempo, eram bastante específicos, ou seja, não se iriam estar a reutilizar. Por outro lado, quando realizava os testes sobre módulos inteiros como a gestão da produção, é muito mais conveniente fazê-lo através de testes manuais, como se precisa de testar sempre os mesmos casos de teste com diferentes *inputs*.

Tabela 2.1 - Vantagens e desvantagens dos testes manuais e automáticos

	Testes manuais	**Testes automáticos**
Vantagens	✓ O *tester* é mais familiar com o produto; ✓ O *tester* consegue escrever casos de teste mais pertinentes; ✓ O *tester* consegue dar feedback mais depressa; ✓ Não é preciso conhecimento de código para os realizar; ✓ Fácil de aprender.	✓ Reutilização de casos de teste e código; ✓ Consistência nos resultados esperados; ✓ A possibilidade de se executar os testes quando se quiser; ✓ Deteta *bugs* antecipadamente em funcionalidades do programa que ainda estiverem em desenvolvimento.
Desvantagens	× É preciso mais tempo que os testes automáticos para realizar, se for preciso realizar vários testes; × Requerem conhecimentos sobre as técnicas de teste; × Algumas funcionalidades do programa podem escapar ao tester; × Os casos de teste não são reutilizáveis.	× Nem sempre é fácil escolher a ferramenta mais apropriada para se testar o *software*; × Ter de se gastar recursos na manutenção dos *scripts* em programas que estão em contínua alteração.

2.2 Black-Box *vs.* White-Box

2.2.1 Black-Box

Em testes de *black-box*, o *tester* não tem informação sobre a parte interna do *software*, o código, enquanto realiza os testes, assim, foca-se no ponto de vista do utilizador para verificar o comportamento do programa. Desta forma, é possível simular o ambiente em que o cliente irá utilizar o *software* e verificar com que erros ele se pode deparar (Loyola-González, 2019).

Existe diferentes formas de implementar este tipo de testagem, sendo as usadas neste projeto:

- Testes funcionais – em que é testado se o *software* se comporta corretamente com o *input* esperado;
- Testes não-funcionais – em que é testado se o *software* se comporta corretamente com o *input* errado, o lado da robustez dos testes não-funcionais, por exemplo, se lança mensagens de erro ou aviso quando o utilizador se engana num *input*;
- Testes de regressão – em que o *software* é testado depois de haver algum tipo de alteração ao código, como correções ou novas funcionalidades adicionadas, para verificar que essas alteraçõesnão alteraram o resto do funcionamento do programa.

2.2.2 White-Box

No *White-box*, o *tester* precisa de ter conhecimento sobre a parte interna do *software*, código, estrutura e arquitetura, para poder analisar o fluxo de operação, a usabilidade e a segurança do programa. Tal é possível através de um conjunto de *inputs* predeterminados para verificar se o resultado dos mesmos são os *outputs* esperados. Um erro é identificado quando um dos *inputs* não resulta num *output* pretendido.

Este tipo de testagem é, normalmente, realizado pelos programadores do *software* em vez de um *tester*, visto que quem tem melhor conhecimento sobre a parte interna do programa é quem a desenvolveu (Loyola-González, 2019). Assim, no contexto do projeto não foram realizados testes *white-box*, estes foram realizados pelos programadores e não pela equipa de *QA*, onde se desenvolveu o projeto.

2.2.3 Vantagens e desvantagens

Apesar das diferenças entre os dois tipos de testagem, um *software* nunca pode ser entregue a clientes sem uma combinação das duas testagens serem realizadas ao programa. Tal porque, um erro pode escapar aos testes de *black-box*, mas ser identificado pelos de *white-box* e vice-versa.

Em *black-box* é possível encontrar *bugs* que os programadores nem sequer tinham pensado em testar, como o programa está a ser testado como um utilizador o utilizaria. Também é possível encontrar erros enquanto o *software* está em desenvolvimento, ajudando a poupar tempo aos programadores, visto que podem continuar o desenvolvimento com os erros identificados em mente. Contudo, muitas das vezes pode-se acabar por gastar mais tempo do que o suposto pois esteve-se a testar um comportamento que não era um requisito da aplicação. Para evitar esta situação, convém a equipa de programadores e *testers* estarem continuamente a partilhar conhecimento entre si.

Por outro lado, com *white-box*, é mais rápido de desenvolver os casos de teste a realizar, como a pessoa que vai testar tem conhecimento sobre a linguagem do código, o que o mesmo tem que fazer e as condições para obter o comportamento correto. Logo, é possível elaborar várias combinações de *inputs* para cobrir grande parte dos requerimentos a testar, o que assegura os clientes que estão a obter um *software* de confiança. No entanto, como só se tem a perspetiva interna do programa, muitas vezes escapam casos de teste que só surgem quando se está a visualizar o programa através do ponto de vista do cliente (Loyola-González, 2019).

2.3 Scripts

Scripts são programas de código que são, normalmente, escritos para serem executados por uma ferramenta de automatização, o que permite executar as tarefas escritas neles todas de seguida, em vez de terem de ser feitas uma de cada vez por um utilizador. Os *scripts* podem ser elaborados em diversas linguagens, como *Python, JavaScript, VBScript, Ruby, C#*, etc. Na maior parte dos casos a linguagem é escolhida tendo em conta que linguagens a ferramenta a ser usada suporta.

Para poderem ser executados do início ao fim sem interrupções, não existe pontos em que o *script* espera algum *input* ou operação do utilizador. Tem de estar tudo definido e configurado antes de o correr, por exemplo, se o *script* vai ler algum ficheiro para realizar uma operação, o ficheiro precisa de ter a operação escrita antes da execução.

Frequentemente, as pessoas que desenvolveram os *scripts* são as que os vão executar, no entanto há casos em que é preciso outras pessoas terem acesso à sua execução. Para tal ser possível, é conveniente os *scripts* serem acompanhados de algum documento que explica o que vai ser executado e as configurações que são precisas antes de se executar.

2.4 TestComplete

A ferramenta de automatização de teste usada no projeto é o *TestComplete*, uma ferramenta especializada na testagem da *interface* de utilizador desenvolvida por *SmartBear* (SmartBear, 2022). *TestComplete* permite testar programas do ponto de vista dos clientes e os programas podem ser de *desktop*, *mobile* e *web*.

TestComplete suporta 7 linguagens de programação como *JavaScript*, *Python*, *C++*, etc. A linguagem que foi usada no projeto foi a *C#*, a linguagem universal usada na empresa *ARTSOFT*, uma vez que, para programadores com bases em *JavaScript, Java,* e *Python*, algumas das linguagens mais usadas (StackOverflow, 2022), é uma linguagem que compartilha vários traços de sintaxe com elas.

O *TestComplete* é usado para executar os *scripts* e validar os seus resultados. Para validar se os resultados obtidos são iguais ao esperado, o *TestComplete* dispõe das opções: imagens; propriedades; ficheiros; bases de dados; objetos; tabelas; *XML* (*Extensible Markup Language*) e web (SmartBear, 2022). Como neste projeto, só foram usadas as validações de imagens, propriedades e ficheiros, são estas que vão ser descritas nas seguintes Subsecções.

2.4.1 Validação - imagens

Na validação através de imagens, é necessário ter um *screenshot* do *software* com o resultado esperado antes de se executar o *script*. Durante a execução, a ferramenta compara a imagem guardada com uma imagem atual do *software*, que o *TestComplete* tira em tempo real. No final da execução, se as imagens forem iguais, a ferramenta diz que o teste foi corrido com sucesso, mas se forem diferentes, o *TestComplete* avisa que as imagens não são iguais e, na ferramenta, demonstra a diferença entre as duas imagens através de uma imagem.

A Figura 2.1, demonstra como o output da ferramenta ocorre. Na parte 1 e 2 da figura estão as imagens que vão ser comparadas com o *TestComplete*, enquanto a parte três é o output. Como podemos ver a imagem que o *TestComplete* gera ignorou a palavra "*Hello*", visto que está escrito nas duas imagens e exatamente no mesmo lugar, contudo, como a palavra "*World*!" só parece numa das imagens, esta a vermelho no output como é a diferença entre as duas.

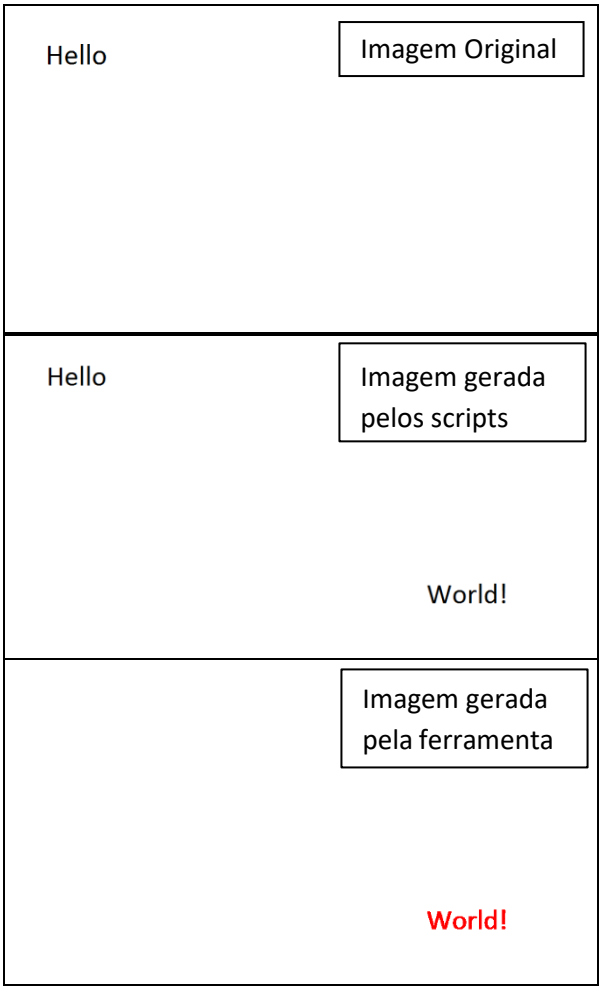

Figura 2.1 Output do TestComplete quando imagens não são iguais

2.4.2 Validação - propriedades

Na validação através de propriedades, no código compara-se o resultado esperado, que é dado como *input*, normalmente através de um ficheiro, com o valor de uma propriedade de um objeto que foi obtido ao executar o *script*. A Figura 2.2 exemplifica o caso de um objeto com várias propriedades, assim, por exemplo, se quisermos saber se o objeto existe escrevemos *object.Exist* e comparamos dentro do código com o valor *boolean true*. Por outro lado, se quisermos verificar que uma propriedade ficou preenchida com um certo valor comparamos *object.wText* com o *input*. Neste projeto não foram usados os separadores *fields*, *methods* e *events* da Figura 2.2.

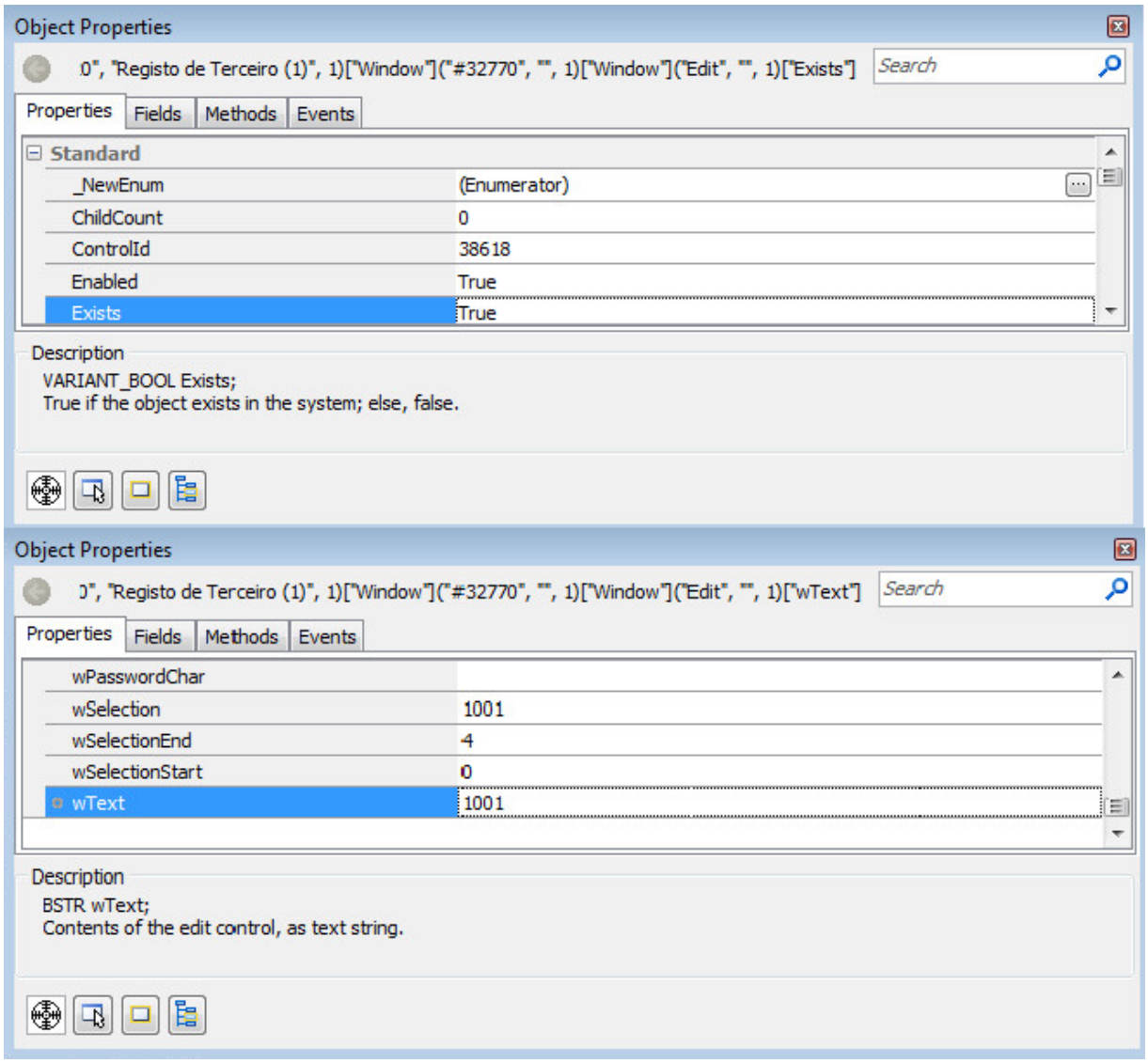

Figura 2.2 Propriedades Exist e wText de um objeto e as suas descrições

2.4.3 Validação - ficheiros

Para validar com ficheiros, é preciso ter o ficheiro com o resultado esperado guardado na pasta do projeto do *TestComplete* antes de se correr o *script*. Durante a execução, a ferramenta compara o ficheiro guardado com o que é gerado pelo *software* em tempo real. Se o *TestComplete* verificar que os ficheiros não são iguais, dá como *output* uma comparação, linha a linha, entre os ficheiros, como verificado na Figura 2.3.

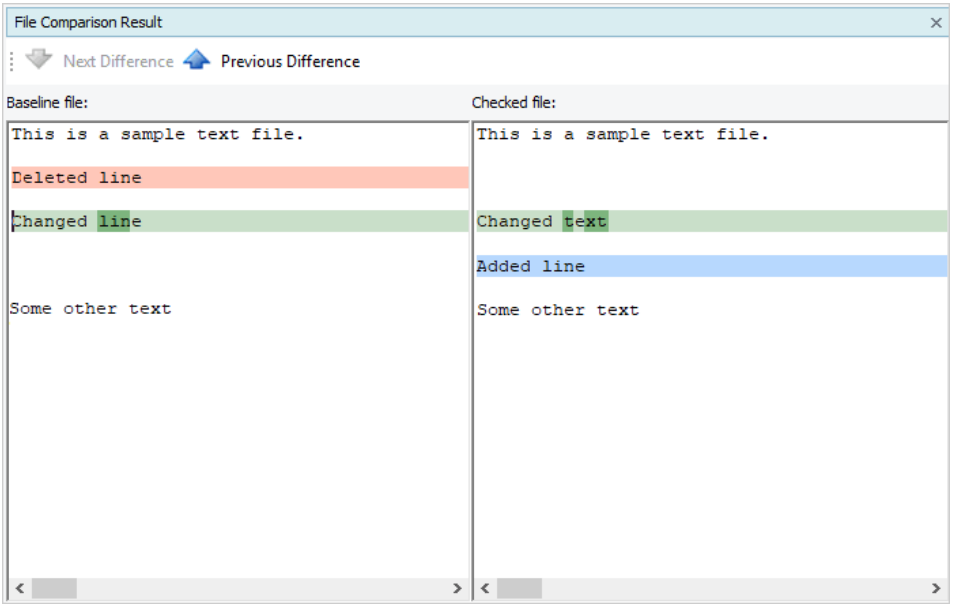

Figura 2.3 Output do TestComplete quando ficheiro não são iguais (SmartBear, 2022)

2.5 Eventos

A empresa *ARTSOFT* utiliza o seu próprio *software* para gerir os seus projetos, logo, a gestão de tarefas de cada trabalhador. Eventos representam tarefas como desenvolvimento de código, correção de *erros*, atendimento ao cliente, prestação de serviço, etc.

2.5.1 Criação de eventos

Na criação de eventos, existe campos para se inserir informações que caraterizam a tarefa a realizar, o que é possível visualizar através da Figura 2.4. Nesta observa-se o que pode ser associado ao evento, como:

- Terceiro - o cliente da empresa a que o evento está associado;
- Evento – o tipo de evento de que se trata, no caso da Figura 2.4 é um evento do tipo *"90.03 Reporte ARTSOFT – Normal"*;
- Assunto – onde se escreve um pequeno resumo do que se trata o evento;
- Texto – uma descrição detalhada de tudo o que envolve o reporte;
- Transferir para – indica o departamento ou o empregado para onde vai ser transferido o evento.

Existe uma pequena grelha onde se encontram os qualificadores:

- Versão *reporte ArtSOFT* – a versão do *software* onde foi identificado o erro;
- Versão correção *ArtSOFT* – a versão do *software* onde o erro vai/foi corrigido;
- *Revision* de correção – o conjunto de versões do *software* onde o erro vai/foi corrigido;
- Módulos *ARTSOFT* – o módulo do *software* em que se encontra o erro;
- Versão – o país da versão do *software* onde foi verificado o erro;

- Reportes – o tipo de erro que está a ser reportado, e.g. não funcionalidade, desenvolvimento, sugestão, entre outros.

Ainda na Figura 2.4 existem dez funcionalidades na coluna localizada na parte esquerda da imagem, sendo os utilizados durante o projeto:

- Suspender – permite fechar o evento e o mesmo continuar atribuído ao mesmo empregado;
- Digitalizar – associa imagens ao evento;
- Visualizar imagens – consultar as imagens associadas através do botão digitalizar.

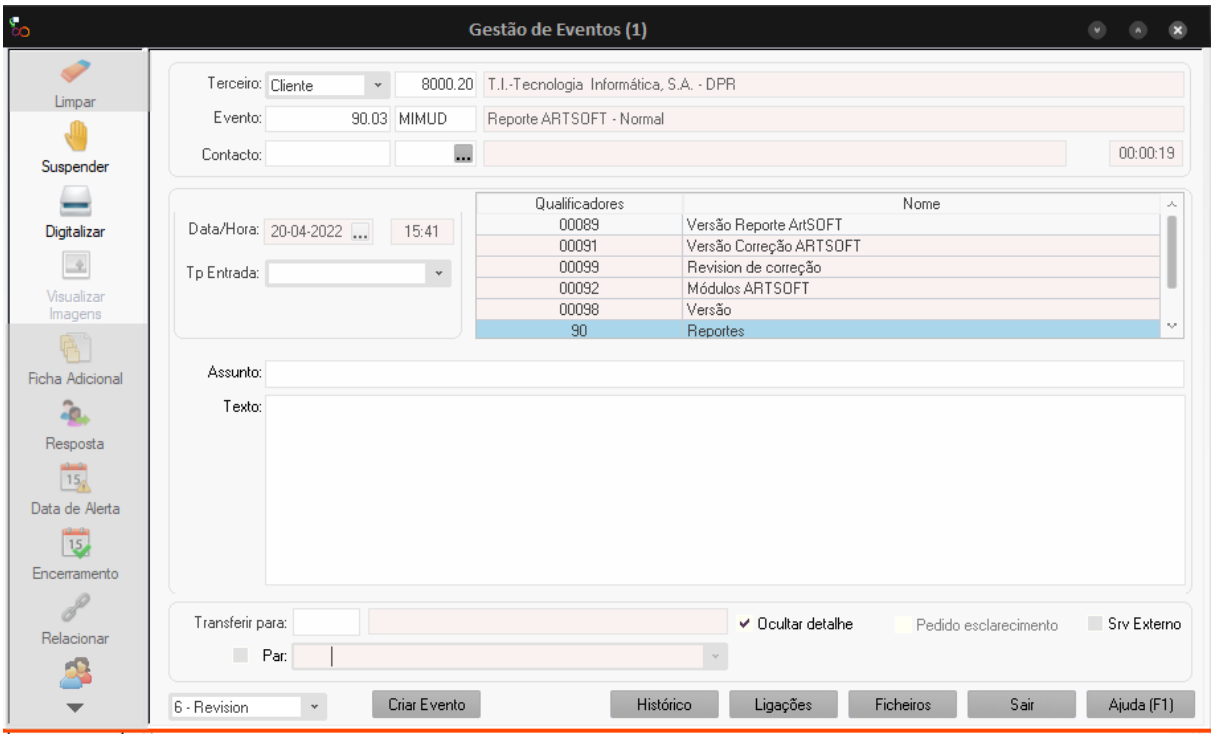

Figura 2.4 Exemplo da criação de um evento

2.5.2 Percurso de eventos

Usando um evento de correção de *bugs para ilustrar* o caminho entre a criação e o encerramento de um evento na empresa: o evento é reportado e transferido para o programador responsável pelo módulo; o programador corrige o erro e transfere o evento para a equipa de *QA* ou diretamente para um *tester*; o *tester* analisa o evento e procede com a testagem; se o *software* tiver o comportamento correto, o evento é transferido para o departamento que tem relação direta com os clientes, caso contrário, é transferido de volta para o programador com a informação do que continua a não ter um comportamento correto e precisa de ser corrigido.

Durante a tese, foram atribuídas funcionalidades ao projeto para testar através destes eventos e os mesmos foram utilizados para reportar erros que verificava durante o meu trabalho.

2.6 Bases de Dados

Uma base de dados é uma ferramenta que armazena informações sobre qualquer tipo de tema, e.g. pessoas, documentos, empresas, veículos, entre outros. No entanto, não se trata só de armazenação, mas também de organização de dados, de forma a não haver entradas repetidas e de ser possível de encontrar um dado específico rapidamente (Microsoft, 2022).

O *ARTSOFT ERP* guarda os seus dados em bases de dados. Cada base de dados representa uma empresa. No ambiente de trabalho do utilizador existe uma pasta com o nome da empresa com a extensão ".art". Esta pasta é a base de dados. Esta pasta é subdividida pelos anos em que a empresa tem dados tal como pastas para guardar documentos relativos a SAFT, SS, entre outros.

As bases de dados guardam informação relativos aos utilizadores, empregados, clientes, configurações, documentos, etc. Logo, as bases de dados são o ambiente de teste com que os *testers* trabalham. Para preparar as bases de dados para a testagem de um evento, o *tester* tem de fazer um *backup* das empresas dos clientes para tentar, ao máximo, replicar a situação que eles reportam. Desta forma, conseguimos com que o nosso ambiente de teste seja o mais parecido ao do cliente possível e que o evento está a ser testado nas melhores condições.

2.7 Módulos do projeto

Como referido em Secções anteriores, o ARTSOFT ERP está dividido em módulos, desta forma uma empresa só precisa de comprar os módulos de que realmente precisa e, por outro lado, torna a gestão do código e da testagem mais fácil.

2.7.1 Workflows

Os *workflows* são máquinas de estado, que são compostas por estados e ações, onde as ações, em conjunto com um *input*, fazem os estados transitarem entre si conforme uma ordem pré-definida.

No *software* da *ARFTSOFT*, os *workflows* seguem o conceito falado em cima, mas aplicam-no a empresas. Assim, o módulo de *workflows* cria fluxos de trabalho que auxiliam a empresa a padronizar os seus processos. Para a empresa saber que processos precisam de ser feitos de forma mais eficaz, deve-se fazer um levantamento da forma como o seu trabalho diário é realizado e analisar o que é feito manualmente e ineficientemente que pode ser transformado num *workflow*.

2.7.1.1 Criação de workflows

A criação de um *workflow* é feita de forma gráfica, de modo que a sua criação seja intuitiva para todos os utilizados, o que significa que a sua construção é feita à base de cliques e *drags*, arrastar de objetos.

Cada estado do *workflow* tem as suas próprias propriedades. É possível definir:

- Os utilizadores que podem apagar e editar a informação do *workflow* quando este se encontra num determinado estado;
- A cor do estado, para que os utilizadores conseguem identificá-lo rapidamente caso algum estado tenha alta prioridade;
- Os utilizadores com acesso a mudar de um estado para o outro;
- Se o estado é um estado inicial ou final;
- Se o estado corresponde à rejeição do *workflow*.

O *workflow* em si também tem propriedades, como os utilizadores com acesso a criar o workflow, o tipo de *workflow* que é, entre outros.

Um exemplo de um *workflow* está representado na Figura 2.5, onde estão definidos quatro estados: pedido efetuado; processar; aceite; rejeitado, e três ações: processar; aprovar; rejeitar, a ligar os estados por ordem. No estado pedido pendente, o estado inicial deste workflow, o utilizador de nome Manuel é o único que pode fazer alterações quando o *workflow* se encontra neste estado. No estado processar, todos os utilizadores que estejam associados ao grupo geral podem realizar alterações. No estado aceite, um dos estados finais do *workflow*, o colaborador com o nome Custódio pode alterar o estado. Por fim,

no estado rejeitado, outro dos estados finais, todos os colaboradores que tenham a função administrador podem efetuar mudanças. Neste *workflow* todos os estados apresentam a cor azul, visto que não há necessidade de destacar um estado dos outros, contudo num *workflow* em que haja um estado de importância elevada, o melhor é defini-lo com cor diferente dos outros para que o utilizador esteja ciente da relevância.

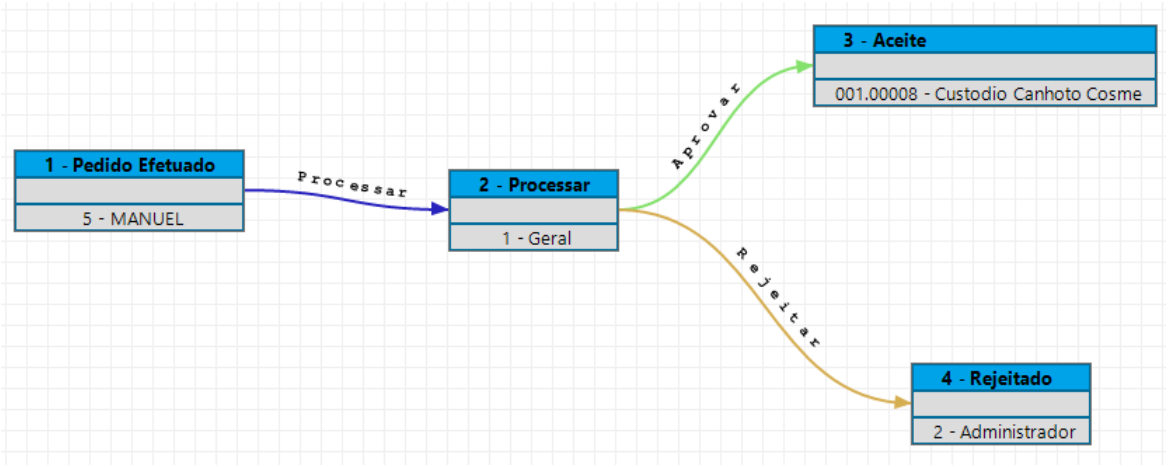

Figura 2.5 Exemplo de um workflow criado no ARTSOFT ERP

2.7.1.2 Tipos de workflow

Os *workflows* podem ser dos seguintes tipos:

- Férias – processo de marcar e aceitar/rejeitar as férias pedidas por um empregado;
- Indisponibilidades – fluxo de marcar a falta de um empregado e aceitar/rejeitar a justificação por detrás dela;
- Presenças – forma de regularizar as horas a que um empregado começou e acabou de trabalhar, isto acontece quando um empregado se esquece de registar as horas de entrada ou de saída;
- Trabalho suplementar – método de um coordenador marcar trabalho fora das horas normais a um empregado e o empregado aceitar/rejeitar;
- Registo de trabalho suplementar – maneira de registar as horas a que o empregado começou e acabou o trabalho suplementar;
- *Workflow* geral – processo pelo qual se pode automatizar qualquer conceito geral pertinente à empresa, por exemplo, atendimento ao cliente.

É possível ter múltiplos *workflows* criados com o mesmo tipo, no entanto, nas configurações das funcionalidades, só se pode associar um *workflow* de cada vez, sendo o tipo *workflow* geral a exceção, visto que não é preciso associar este tipo a uma funcionalidade. A Figura 2.6 demonstra esta situação, onde a funcionalidade de férias está associada ao *workflow* de código dois e por assim em diante.

Figura 2.6 Exemplo de configurações onde se associam workflows a funcionalidades

2.7.2 Produção

O módulo de produção tem como função o controlo e gestão dos processos da indústria produtiva, o que inclui uma monitorização em tempo real para que as empresas conseguiam otimizar os seus custos e stocks. Isto devido ao facto de a indústria de produção de produtos está em constantes alterações diariamente, o que torna difícil a gestão das empresas deste setor sem um *software* de *ERP* para fazer uma gestão eficaz dos seus recursos.

2.7.2.1 Ficha técnica

A ficha técnica define a relação existente entre o produto final e os restantes componentes intermédios necessários à sua produção. Os componentes intermédios têm de ter uma ficha técnica de produto acabado associada.

A ficha técnica apresenta as tarefas que representam os processos realizados durante uma fase e as regras necessárias para a produção do produto final pedido. Logo, uma tarefa tem de estar sempre associada a uma fase. Como as tarefas têm estas informações são importantes no planeamento de encomendas, visto que este tem em conta a duração das tarefas e a alocação de recursos.

O separador de ordens de fabrico apresenta todas as ordens de fabrico criadas que estão associadas à ficha técnica, podendo aceder a cada uma através desta janela.

2.7.2.2 Ordens de fabrico

A ordem de fabrico junta todas as informações do que a empresa produziu e como produziu e, assim, demonstra toda a informação que está espalhada pelas várias áreas do módulo de produção.

Quando é criada uma ordem de fabrico é necessário alterar a ficha técnica referente ao produto final da ordem de fabrico, visto que as mudanças só afetam a ordem enquanto a ficha técnica continua igual. O mesmo acontece para as fases e tarefas.

Na Figura 2.7 é possível visualizar os seguintes elementos:

- Pré reserva – gera documentos de produção com os respetivos custos de materiais, consumos e produtos finais;
- Prioridade – atribui uma prioridade à ordem de fabrico para destacá-las das outras;

- Progresso – demonstra a evolução da ordem de fabrico.

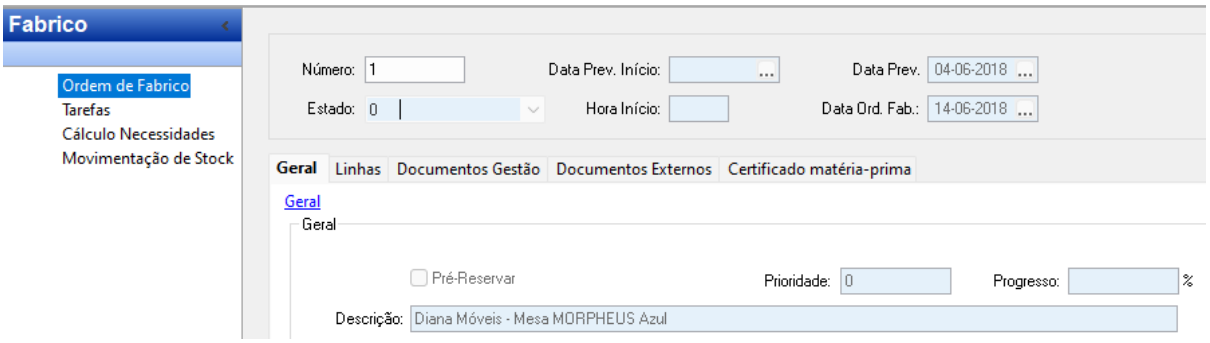

Figura 2.7 Exemplo de uma ordem de fabrico

Nas ordens de fabrico os utilizadores conseguem anexar ficheiros relativos a elas, como imagens ou documentos, de modo a ficarem organizados e acessíveis. Para além, destes ficheiros, é possível adicionar certificados dos materiais usados, associando-os às linhas da ordem de fabrico. Assim, os produtos finais ficam com a indicação dos certificados dos materiais usados na sua produção.

A ordem de fabrico, ao evoluir de estado, permite registar o que foi trabalhado, durante quanto tempo e por quem. Estes registos servem para as empresas conseguirem fazer uma análise sobre o trabalho realizado e verificar se foi executado como se tinha planeado no início do processo, de forma a poderem atualizar os seus processos e conseguirem produzir de forma mais eficaz.

As movimentações de *stock* dos materiais são guardadas nas ordens de fabrico. Para criar uma movimentação de *stock* é preciso que tenham sido geradas pré reservas, uma vez que a movimentação transforma as pré reservas em movimentos efetivos de *stock*. Uma vez transformado, um movimento não pode ser retificado, mas pode ser transferido outra vez para a pré reserva.

Ao visualizar os movimentos, é possível observar cada movimento efetuado dos vários artigos associados à ordem de fabrico.

2.7.2.3 Gestão operacional

Na gestão operacional o cliente consegue visualizar os documentos e fases que foram criadas pelas ordens de fabrico. Estas são apresentadas, num calendário, com as tarefas que estão associadas às mesmas e as suas linhas. Também é possível visualizar num cronograma, com uma linha temporal, como estão distribuídas as tarefas por linha da ordem de fabrico.

2.7.2.4 Planeamento

O planeamento é o que organiza a informação definida na ficha técnica. Com esta informação é possível fazer o cálculo das necessidades para fabrico tendo em conta as encomendas pedidas à empresa e quando é que têm de ser entregues. Para efetuar o cálculo é preciso marcar as encomendas que se pretendem produzir e, assim, a empresa fica a saber desde o início o que é preciso e o custo da produção que lhe foi pedida.

É no planeamento que são geradas as ordens de fabrico, como uma ordem de fabrico é baseada no cálculo de necessidade de produção e as encomendas definidas num documento de encomendas, estas encomendas podem ser encomendas clientes ou interna.

Nos cálculos de necessidades é tido em conta:

- Quantidade requisitada – a quantidade requisitada do produto;

- Quantidade encomendada – a quantidade que foi solicitada a fornecedores de materiais;
- Quantidade em produção – a quantidade do produto final que se encontra, em tempo real, em produção;
- Quantidade em armazém – a quantidade do produto final e de materiais que se encontram em *stock* em armazéns;
- Quantidade necessária – a quantidade que é preciso produzir.

2.7.3 Assistências técnicas

As assistências técnicas correspondem a serviços pós-venda disponibilizados por empresas como, oficinas, reparação de eletrodomésticos, instalação e certificação de canalizações, assistência de *software*, entre outras. Nos serviços de pós-venda, existe um circuito onde estão envolvidos vários empregados e equipamentos, desde o pedido de assistência até à reparação e entrega do equipamento a funcionar corretamente.

No *ARTSOFT ERP*, os recursos necessários para os serviços são contabilizados desde o início do processo, como os equipamentos que precisão de reparação ou são para substituir, os custos de deslocação dos empregados e as informações dos clientes. Os empregados da empresa encarregues dos equipamentos têm a possibilidade de acompanhar, em tempo real, os estados dos processos, de forma a ter conhecimento do que está a acontecer com os equipamentos e saber quais as tarefas que precisam de ser realizadas e as suas datas-limite.

2.7.3.1 Equipamento

Os equipamentos são registados na base de dados do *ERP*, através da criação do seu próprio registo. É no registo onde estão guardadas todas as descrições pertinentes ao equipamento:

- Descrição - onde é dado um nome ao equipamento para o conseguir identificar mais facilmente;
- Marca - o nome da empresa que produziu o equipamento;
- Modelo - o nome do modelo do equipamento;
- Observações - um campo que permite escrever qualquer detalhe sobre o equipamento que não pertence a nenhum campo do registo;
- Imagem - que mostra uma imagem representante do equipamento que foi associada ao registo pela empresa;
- Proprietário - onde consta os detalhes do cliente a que pertence o equipamento tal como o histórico de todos os clientes que já foram proprietários do equipamento desde que esteve registado na empresa;
- Localização - que pode ser a localização do proprietário ou outro lugar como um armazém;
- Garantia - que contém o representante e o tipo de garantia do equipamento;
- Seguro - que descreve a seguradora, o mediador, a apólice, e a data-limite do seguro do equipamento;
- Contrato - caso o equipamento esteja sobre o abrigo de algum contrato em vigor que foi estabelecido entre o cliente e a empresa de assistência estes campos descreve os detalhes do mesmo;
- Imobilizado - caso o equipamento pertença ao património da empresa de assistência, neste campo a empresa pode associá-lo a uma ficha de bem e a um ativo associado;
- Data de abate - se o equipamento não tiver reparação possível, regista-se neste campo a data em que foi abatido;

- Documentação associada - onde se associa quaisquer documentos relativos ao equipamento, como manuais ou comprovativos, para se poder os ficheiros estejam organizados para consultados quando necessário.

A Figura 2.8 demonstra o exemplo de um equipamento, onde é possível visualizar alguns dos campos descritos anteriormente.

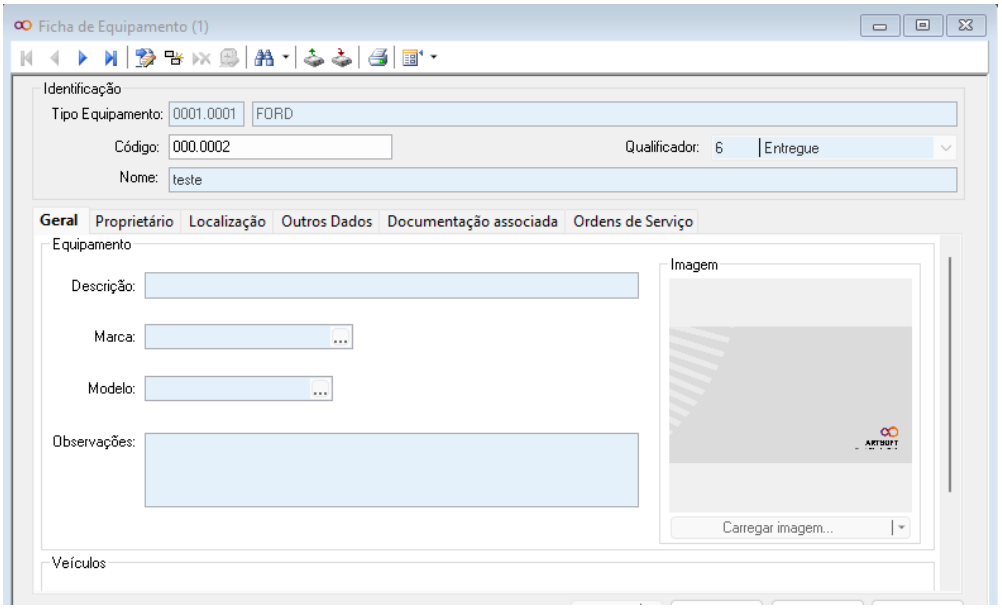

Figura 2.8 Exemplo de um equipamento

2.7.3.2 Ordens de serviço

As ordens de serviço contêm uma lista de tarefas que precisam de ser realizadas para o equipamento estar pronto para o cliente e uma lista de artigos que vão ser necessários requisitar para resolver a ordem. Cada tarefa tem um tempo estimado de execução, para a empresa conseguir controlar o prazo de entrega do produto e quando cada tarefa tem de ser concluída.

Os artigos usados numa ordem têm de ser requisitados, usados e faturados. Para tal existem os seguintes campos para organizar os artigos, gerar documentos de requisição e faturação automaticamente e verificar o *stock* de cada artigo:

- Quantidade faturável – uma quantidade de artigo que é configurada na ordem de serviço, contudo é possível alterar antes de se requisitar os artigos para se poder gerar a fatura com a quantidade correta;
- Quantidade já utilizada – um campo preenchido automaticamente pela quantidade requisitada de artigos e mão de obra;
- Quantidade a requisitar – a quantidade que é preciso requisitar de cada artigo para se poder realizar o serviço sobre o equipamento;
- Armazém – o número do armazém para onde os artigos requisitados vão ficar guardados;
- *Stock* efetivo – a quantidade de cada artigo que está guardado no armazém indicado.

Ao requisitar artigos ou ao encerrar a ordem, a mesma gera documentos com informações de orçamentos ou faturas. Assim, a ordem guarda num separador todos os ficheiros que foram gerados por ela.

Para além dos documentos gerados, é possível guardar na ordem de serviço, documentos relativos ao equipamento associado, às tarefas, aos materiais, etc. Deste modo, é possível consultar os documentos de forma eficaz.

A ordem de serviço também guarda um histórico de cada vez que o estado dela é alterado, quem a alterou e quando, para estar em cima de se a ordem está dentro dos prazos conforme vai mudando de estado.

Na Figura 2.9, é possível visualizar uma ordem de serviço de uma revisão dos 15.000 Km do equipamento de nome teste, sendo o proprietário o cliente de código 1001. Na grelha do separador aberto na imagem, verificamos as tarefas que são necessárias realizar, os artigos que os trabalhadores vão precisar, a duração e, depois de ser planeada, irá aparecer as datas de início e fim das tarefas. Na parte de baixo da janela existe o botão "Gerar Doc." que permite gerar os documentos de requisição, orçamento, faturação, entre outros.

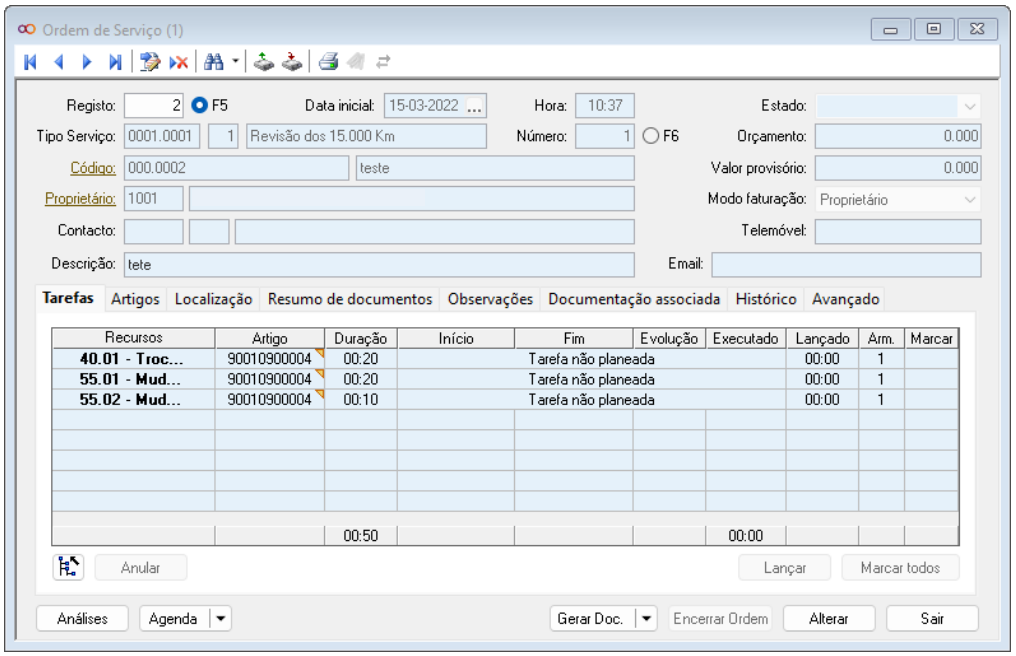

Figura 2.9 Exemplo de uma ordem de serviço

2.7.3.3 Orçamento

O orçamento é bastante parecido com a ordem de serviço, tem exatamente os mesmos campos e informação que a ordem tem. Podemos chamar ao orçamento uma pré ordem de serviço, uma vez que o orçamento é feito para enviar aos clientes os detalhes todos de como a sua assistência vai decorrer e quanto é que vai custar, em tempo e em custo.

Se o cliente aprovar o orçamente desenvolvido, então usasse um botão chamado "Gerar Ordem" que vai criar uma ordem de serviço com todas as informações do orçamento transferidas para ela. Desta forma, é descomplicado o processo de criar uma simulação de uma ordem de serviço para dar um orçamento ao cliente e depois transformá-la numa ordem de serviço verdadeira.

3 Metodologia

Com o objetivo de o projeto ser testar a fundo as funcionalidades do *ARTSOFT ERP*: *workflows*; assistências técnicas e produção, de modo que o *software* entregue ao cliente seja um de confiança e sem erros, foram implementadas diferentes técnicas e tipos de testagem.

3.1 Eventos

Os eventos representam modificações realizadas ao *software*, como correções ou criação de novas funcionalidades, tal como mencionado na Subsecção 2.5.

Ao longo do projeto, foram atribuídos eventos diariamente para proceder à testagem das alterações feitas. A testagem foi elaborada através de testes manuais, onde , como explicado na Subsecção 2.1.1, é necessário ter conhecimentos profundos sobre a funcionalidade sobre a qual se está a testar. Desta forma, se quando começa a analisar um evento, para além de se analisar o contexto do mesmo, é necessário ir à documentação do programa para estudar qual o comportamento esperado do *software* tal como que outras funcionalidades podem ter sido afetadas com as alterações efetuadas. À medida que se vai analisando a documentação, num ficheiro à parte, anota-se os casos de teste a realizar.

Após ter os casos de teste descritos, é preciso ter o ambiente de teste preparado para os executar. Isto envolve preparar uma base de dados, com as configurações e os critérios que são indicados no evento. Neste projeto, os meus ambientes de testes consistiram em cópias das bases de dados dos clientes, para ter a certeza de que estou a simular os passos que os futuros utilizadores iram seguir.

Com os casos de teste e o ambiente de teste preparados, executam-se os testes. No caso de eventos, os testes elaborados foram do tipo *black-box*, que, como descrito na Subsecção 2.2.1, são testes realizados do ponto de vista do cliente sem o *tester* ter acesso à parte interna do *software*. No fim da testagem, registam-se os resultados obtidos e comparam-se com os esperados para verificar que o programa está a funcionar corretamente.

Quando dos testes se obtêm resultados corretos é dado como concluído a tarefa de testagem e transfere-se o evento para que as alterações possam ser disponibilizadas aos clientes nas próximas versões do programa. Por outro lado, se os resultados esperados forem diferentes dos obtidos é preciso rever a suposta correção ou alteração feita ao código. Para isso o evento é transferido para o programador que realizou as modificações para que possam verificar a situação e corrigir.

No entanto, nem todos os eventos seguem os dois processos descritos em cima. Em alguns eventos podemos nos deparar com situações duvidosas, como possíveis erros para além do que foi descrito no evento. Neste caso, é preciso realizar uma reunião com o programador encarregue do evento e discute-se como proceder, se o comportamento do *software* é correto ou se é necessário alterar o código atual. Se da reunião, sair a conclusão de que o *ARTSOFT* precisa de corrigido, é da responsabilidade do *tester* de criar um evento a reportar o *bug* verificado durante a reunião para que a equipa de programação possa analisar a situação.

3.2 Dados

Como descrito na Subsecção 2.6, os ambientes de teste usados durante o projeto foram bases de dados que representam empresas. Desta forma, foi preciso criar uma base de dados para cada script: workflows; assistências técnicas; produção. Cada uma com as suas próprias configurações associadas aos módulos.

3.2.1 Configurações de *workflows*

Na empresa usada para executar os *scripts* de *workflows*, foram configurados os seguintes conceitos:

- 7 Empregados - cada um com o seu departamento, horário, renumerações definidas e função de colaborador;
- 5 Utilizadores - estes vão ser as pessoas que vão executar os casos de teste. Só têm acesso ao módulo de *workflows*, assiduidades e projetos, evitando mudanças no resto do programa, e mesmo nestes só têm acessos normais, ou seja, não têm acessos de supervisor[1]. Visto que, o acesso supervisor, dar-lhe-ia permissão para mudar um *workflow* para qualquer estado e, assim, pôr em causa os resultados obtidos durante a testagem. Cada utilizador está integrado num grupo;
- 2 Funções - 1) colaborador, onde estão associados todos os empregados da empresa, e 2) administrador, composto por três utilizadores que coordenam o resto dos empregados;
- 2 Grupos - 1) chefia, constituído por um utilizador que representa o departamento com controlo sobre a empresa, e 2) recursos humanos, formado por um utilizador que representa o departamento que lida com os assuntos de assiduidades;
- Presenças - é preciso estar geradas presenças para um empregado antes da testagem, ou seja, um empregado num dia, precisa de realizar três registos de ponto, indicação de quanto começou/acabou o trabalho. Como uma presença, para ser regular, precisa de ter quatro registos de ponto vai ser gerada uma presença por regularizar. É esta que vai servir para a testagem do tipo de *workflow* presenças.

3.2.2 Configurações de assistências técnicas

Na empresa usada para executar os *scripts* de assistências técnicas, foram configurados os seguintes campos:

- 3 Tipos de equipamento – cada um com vários subtipos, como demonstrado na Figura 3.1;
- 6 Qualificadores de equipamento – que informam em que estado o equipamento se encontra;
- 2 Marcas/Modelos de equipamento – que indica as marcas/modelos disponíveis para definir um equipamento;
- 7 Tipos de artigos para as ordens de serviço – cada tipo tem os códigos dos artigos que se encontram na categoria;
- 3 Secções de documentos das ordens de serviço – que define os documentos que as ordens de serviço conseguem gerar;
- 3 Tipos de ordens de serviço – cada um com diferentes subtipos;
- 6 Estados das ordens de serviço – que indica em que passo a ordem de serviço se encontra, por exemplo, em diagnóstico ou encerrada;
- 5 Tipos de tarefas – em que cada tipo tem tarefas (Secção 2.7.3.2) que se encontram dentro da categoria e vão poder ser usadas nas ordens de serviço cujo tipo necessite delas;
- 3 Estados de tarefas – indica em que passo está a tarefa, por iniciar, iniciado ou terminado.

Na Figura 3.1, é possível observar como os tipos de equipamentos são formados. Existe um tipo principal, neste caso "Veículos", e um subtipo do principal, na Figura existem 4: FORD; OPEL; PEUGEOT; RENAULT. É possível navegar pelos tipos através das setas e também ir diretamente para o primeiro e o último tipo.

[1] Neste projeto supervisor e administrador não são sinónimos. Em termos de hierarquia o supervisor é superior ao administrador, ou seja, tem acesso a mais funcionalidades.

Para apagar os tipos, pode se apagar cada linha da tabela diretamente apagar a tabela inteira com o tipo e os seus subtipos, através do botão de cruz vermelha.

Os outros separadores do lado esquerdo da Figura 3.1, cujos conceitos já foram explicados anteriormente, funcionam da mesma forma que este exemplo dos tipos de equipamentos.

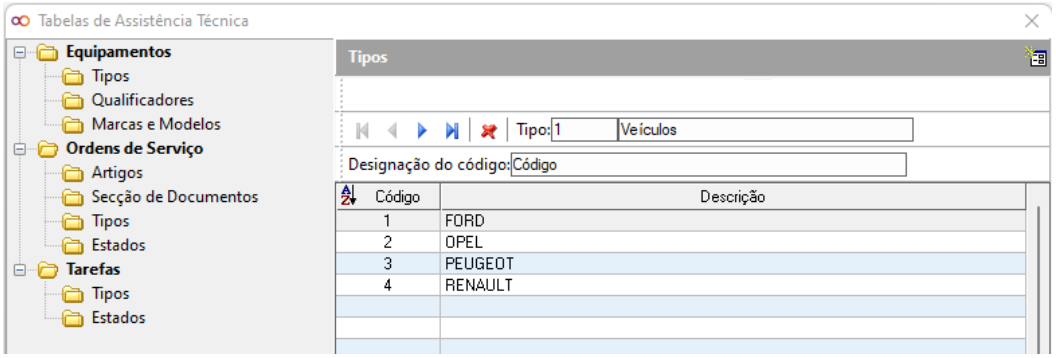

Figura 3.1 Configurações de assistências técnicas

3.2.3 Configurações de produção

Na empresa usada para executar os *scripts* de produção, foram configurados os seguintes campos:

- 2 Tipos de ordens de fabrico – que definem a categoria do produto a fabricar, sendo um exemplo a Figura 3.2;
- 5 Estados de ordens de fabrico – indicam a situação da ordem, se está em preparação, parada ou concluída, por exemplo;
- 5 Estados de linhas de ordens de fabrico – a mesma coisa que os estados das ordens de fabrico, mas neste caso para as suas linhas.

Na Figura 3.2, no lado esquerdo é possível verificar três diferentes separadores: “tipos”; “estados ordem fabrico” e “estados linha ordem fabrico”, sendo o separador que está aberto na janela principal o “tipos”. Todos os separadores têm a mesma configuração, uma linha com botões e uma tabela em que se insere o número e a descrição do tipo/estado. Os botões disponíveis são, da esquerda para a direita:

- Inserir novo registo – insere na grelha um novo tipo/estado;
- Alterar registo – permite fazer alterações sobre um tipo/estado já criado;
- Apagar registo – apaga o tipo/estado selecionado;
- Pesquisar texto – permite encontrar um tipo/estado através de palavras;
- Listar tabela – imprimi as informações da tabela;
- Importar dados para a tabela – importa dados de outra tabela para a tabela aberta;
- Exportar dados da tabela – exporta os dados da tabela e guarda-os;
- Ajuda – permite ao utilizador saber como funciona a janela;
- Sair – fecha a configuração.

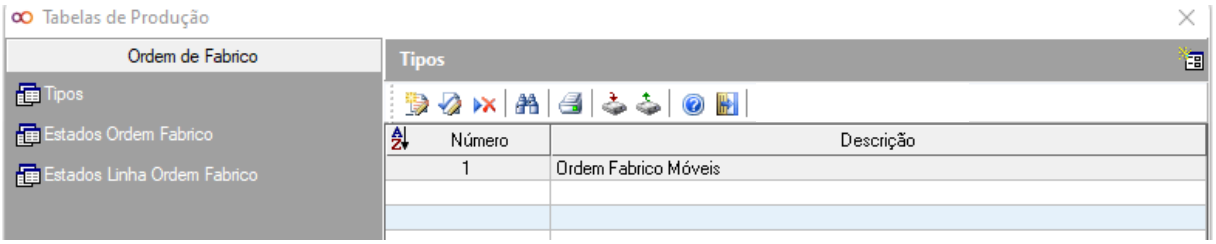

Figura 3.2 Configurações de gestão de produção

3.3 Planos de teste

O documento de plano de testes é uma parte fundamental parte do processo de testagem automática, como referido na Subsecção 2.1.2, visto que é neste que são definidas as funcionalidades a testar e os casos de testes que vão ser executados.

Para elaborar o documento é preciso estudar detalhadamente os manuais do *software* para ter conhecimentos das funcionalidades atualmente disponíveis nos módulos a testar e o comportamento das mesmas. O estudo do programa é acompanhado com algumas reuniões com o programador que desenvolveu o módulo ou funcionalidade para tirar dúvidas que tenham surgido, assim, o tester recebe o conhecimento mais atualizado possível.

Seguidamente, começa-se a elaborar o documento, com as descrições das funcionalidades e módulos a testar e desenvolvem-se os casos de teste, como demonstrado na Figura 3.3, onde estão descritas as pré e pós condições de cada caso de teste, seguido de cada passo a seguir e os seus *inputs*, atalhos de teclado e resultados esperados.

11. WORKFLOW ASSIDUIDADES – FÉRIAS

Pré-condições: workflow férias criado e ter o utilizador 2 como coordenador do 3.

Pós-condições: féria cancelada, na 21.1, e aprovada, na 21.0.

Operação	Input	Atalho	Resultado Esperado
Login com o user 3	3|	Click	3 é o user
Ir a Assiduidades-Colaborador Interno-Férias	Assiduidades.Ferias	[Ctrl]+[Alt]+[Q]	Janela de férias aberta
Criar		ClickButton	Janela de criação aberta
Colaborador: ele próprio		[+]+[Down]+[Enter]	Colaborador escolhido
Data início: 16-02-2022	16-02-2022		Data escrita
Data fim: 16-02-2022	16-02-2022		Data escrita
Gravar		ClickButton	Janela fecha e workflow grava
Validar colaborador		Property	
Validar data início		Property	
Validar data fim		Property	
Login com o user 2	2|	Click	2 é o user
Ações: Processado	Processado	[Tab] ou Click+[Down]*	Estado muda para "Processado"
Validar estado		Property	
Validar estado anterior		Property	
Na 21.1 Login com o user 3	3|	Click	3 é o user
Na 21.1 Ações: Cancelado	Cancelado	[Tab] ou Click+[Down]*	Estado muda para "Cancelado"
Na 21.0 Ações: Aprovado		[Tab] ou Click+[Down]*	Estado muda para "Aprovado"
Validar estado		Property	
Validar estado anterior		Property	

Figura 3.3 Exemplo de um caso de teste com cada passo e resultado esperado descrito

3.4 Scripts

Após a conclusão do plano de testes e da consequente validação pela equipa, começa-se a traduzir o os casos de teste para código que o *TestComplete* pode executar, neste projeto foi usada a *C#*, a linguagem usada internamente na *ARTSOFT*.

Enquanto se está a elaborar os *scripts* convém ter um *back-up*da base de dados antes de se executar uma operação sobre ela, desta forma, os resultados obtidos são confiáveis.

Quando se escreve em código um caso de teste é preciso executá-lo para verificar se está a correr sem erros ou *warnings*, um aviso de que uma validação entre resultados obtidos e esperados não está correta. Após o caso correr sem problemas procede-se à escrita do próximo caso. No final, procede-se à execução de todos os casos de teste em conjunto para verificar se alguns interferem entre si e para analisar o tempo que o *script* demora a correr para indicar aos supervisores, com o objetivo de se poder programar a execução dos *scripts* atempadamente a um lançamento de uma nova versão aos clientes.

Como referido, os *scripts* são executados sobre uma versão nova quando esta está para ser entregue aos clientes. Se, durante a execução, a ferramenta lançar um *warning*, o que significa que o *software* não

está a ter o comportamento correto, tem de ser criado um evento a reportar o erro verificado e transferi-lo para a equipa de programação responsável com um alto nível de importância para que possa ser resolvido a tempo do lançamento da nova versão. Quando os programadores corrigirem o *bug*, volta-se a executar os *scripts* para verificar que não existem mais e que a versão está pronta para os clientes.

Os *scripts* realizados durante o projeto recebem *inputs* através de ficheiros, onde se escrevem os valores que foram descritos na coluna "*Input*" da Figura 3.3. Assim, a primeira operação que se realiza nos *scripts* é abrir e ler o ficheiro que foi indicado como *input*. Cada linha do ficheiro corresponde a um caso de teste, logo o *script* acaba a sua execução quando se chega à última linha do ficheiro.

4 Resultados

Com o trabalho desenvolvido durante o projeto, foram obtidos resultados sobre a qualidade dos módulos de *workflows*, assistência técnica e produção tal como o *ARTSOFT ERP* como um todo.

Em todos os módulos, os *scripts* implementados foram mantidos durante todo o tempo do projeto, de modo a funcionarem em todas as versões lançadas do software, tal como sempre bem documentados, através de planos de teste bastante descritivos e detalhados e comentários ao longo do código para se conseguir transmitir o raciocínio por detrás das linhas de código escritas.

Em termos, de o que foi planeado desenvolver neste projeto e o que, na prática, foi desenvolvido, não houve diferenças. Todos os módulos e eventos que eram supostos serem testados foram testados, resultando numa eficácia de 100% nas tarefas deste projeto.

4.1.1 Resultados do módulo de workflows

Relativamente ao módulo de *workflows*, com os *scripts* que foram desenvolvidos e executados sobre as versões do *ERP* lançadas entre novembro e abril, foi verificado comportamento errado do *ARTSFOT* em várias situações. Para o desenvolvimento inicial do plano de teste foi necessário 12 dias mais um mês para escrever os scripts. Consequentemente, foram reportados um total de 42 erros, do qual 40 deles foram corrigidos e resolvidos, e 1 sugestão de melhoria relativamente à interface de criação de um *workflow*, a qual foi analisada e verificada ser pertinente para adicionada ao programa, pois irá fazer a manutenção dos *scripts* demorar menos tempo tal como ser mais fácil de utilizar para os clientes.

Nos *scripts* desenvolvidos foram feitos 33 casos de teste. Desta forma, conseguiu-se verificar todas as funcionalidades do módulo e, consequentemente, preveniu-se os clientes de encontrarem erros durante a sua utilização.

4.1.2 Resultados do módulo de produção

No desenvolvimento foi gasto um mês e 12 dias na escrita do plano de teste mais 15 dias para a implementação dos respetivos scripts. Os *scripts* do módulo de produção foram executados sobre as versões do *software* lançadas entre janeiro e abril, onde foram reportados um total de 32 *bugs*, dos quais todos foram corrigidos pela equipa de programação e voltados a ser testados pela equipa de *QA*.

Nos *scripts* foram desenvolvidos 57 casos de teste, devido à grande dimensão do módulo, para que nenhuma funcionalidade ficasse fora do *scope* de teste.

4.1.3 Resultados do módulo de assistências técnicas

No desenvolvimento foi gasto um 10 na escrita do plano de teste mais 7 dias para a implementação dos respetivos scripts. Os *scripts* do módulo de assistências técnicas foram executados sobre as versões do *software* lançadas entre março e abril, onde foram reportados um total de 24 bugs, dos quais 18 foram corrigidos pelo programador encarregue.

Nos *scripts* foram desenvolvidos 221 casos de teste, o maior número de casos de teste de todos os *scripts* devido a um número de ligações a outros módulos, como a gestão comercial.

4.1.4 Resultados dos testes manuais

No resto de *software*, foram realizados testes manuais, a partir de eventos, à medida que *bugs* eram corrigidos e novas funcionalidades adicionadas. Estes testes resultaram num total de 405 eventos testados e resolvidos. Dentre estes eventos, 13% tiveram de ser transferidos de volta para o programador encarregue para resolverem de novo a situação.

A Figura 4.1 demonstra um dia de trabalho na empresa, em que os retângulos vermelhos representam eventos e a sua largura, o tempo que se demorou a tratar de cada um. Há eventos que que estão sobrepostos, estes são casos os que, durante a testagem de um evento, foram encontrados *bugs* que não o reportado e, logo, foi criado outro evento para que o erro fosse resolvido.

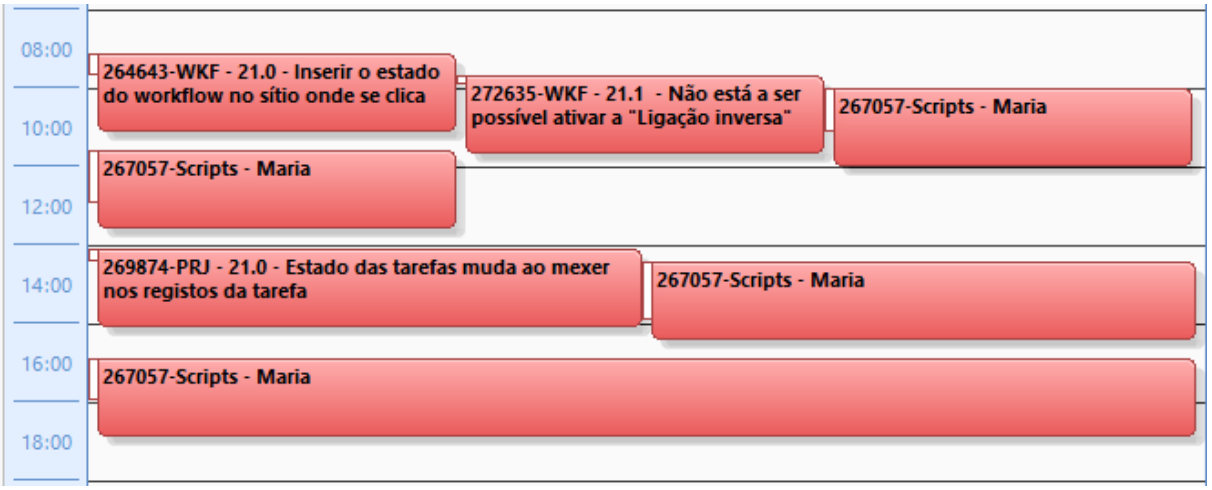

Figura 4.1 Calendário de um dia de trabalho com eventos alocados

5 Discussão

Os resultados obtidos foram frutos de trabalho diário e contínuo na empresa, tendo feito em média quatro eventos por dia, o que corresponde a vários casos de teste realizados por cada evento. Uma vantagem a que esta testagem diária traz é encontrar erros no *software* que, caso contrário, poderiam passar despercebidos, o que sublinha a importância que os testes têm na manutenção da qualidade do *software*.

Assim, o resultado do presente trabalho demonstra a importância de ter uma equipa de programadores dedicada a *Quality Assurance*, devido ao facto de:

- Ser possível encontrar um *bug* no início do desenvolvimento de uma nova funcionalidade e, assim, ser logo corrigido. Isto faz com que aumente a eficiência da empresa, ao não ter de corrigir o erro no final do desenvolvimento onde se vai ter de alterar muito mais código do que no início. No caso deste projeto, com os erros reportados a partir dos *scripts* desenvolvidos, os descritos na Secção 4, foi possível aumentar a eficácia da empresa, visto que todos os módulos eram de desenvolvimento recente;
- Poder dar a garantia aos clientes de que os requisitos por que eles estão a pagar funcionam a 100%. É difícil chegar a esta percentagem, mas com a ajuda da testagem automática realizada é possível aproximar-se ao máximo. Os testes são escritos do ponto de vista do cliente, desta forma, trabalhamos com o *input* que utilizadores vão usar na aplicação juntamente com o nosso conhecimento dos pré-requisitos de cada funcionalidade;
- Ter clientes satisfeitos, se não houvesse os *scripts* a detetarem os *bugs* na Secção 4, eles iriam passar para a versão final do *ARTSOFT ERP*. Assim, quem iria verificar os erros eram os clientes no meio do uso da aplicação, o que resulta em utilizadores frustrados porque não conseguem realizar o seu trabalho e, consequentemente, com menos confiança no programa e com mais probabilidade de irem ver a concorrência;
- Mais cenários são tidos em conta durante o desenvolvimento. Com a equipa de *QA* a trabalhar lado a lado com os programadores do programa, os *testers* podem dar *feedback*, de um ponto de vista de um utilizador, de o que pode estar a faltar numa funcionalidade, por exemplo, alguma explicação ou proteção contra erros.
- Manutenção do código de forma mais eficaz, com os *scripts* já desenvolvidos, sempre que há alguma alteração no código, para verificar que tudo está a trabalhar conforme os requisitos basta executar os *scripts*. Se houver algum erro, é possível identificar a parte do código que precisa de correção ao verificar qual o caso de teste que falhou.

Outro fator que é concluído através dos resultados é o tempo gasto em testes e se o mesmo vale a pena os recursos usados.

Entre os dois tipos de testes desenvolvidos, manuais e automáticos, o automático demora mais tempo a desenvolver, uma vez que, os manuais são usados em testes de eventos, ou seja, em que não é preciso testar muitas funcionalidades. Por outro lados, os automáticos são usados para testar módulos inteiros com várias funcionalidades interligadas entre si, o que corresponde a muitos mais casos de teste.

No entanto, quando se trata de tempo de execução, os testes automáticos são mais rápidos e práticos, uma vez que basta executar os *scripts* e, enquanto eles estão a correr, pode-se ir tratando de outras tarefas.

Para além dos testes, também vale a pena discutir o tempo gasto em cada módulo testado. Entre o *workflow*, produção e assistências técnicas, o que levou mais tempo foi o módulo de produção, isto porque é o que tem mais detalhes e interligações a outras funcionalidades fora do módulo o que leva a configurações bastante complicadas e que, infelizmente, nem sempre estão totalmente documentadas e é preciso ir falar com a equipa de programação.

Contudo, apesar de ter sido o módulo em que se gastou mais tempo neste projeto não quer dizer que seja o mais importante, ou seja, o mais usado pelos clientes, no caso deste projeto são os módulos de produção e assistências técnicas.

Os módulos de produção e de assistências técnicas estão empatados quanto à sua importância, visto que a *ARTSOFT* tem bastantes clientes que fornecem serviços pós-venda e que fabricam os seus próprios produtos. Assim, pode-se inferir que o esforço gasto a elaborar e executar cada caso de teste nestes módulos foi eficaz, pois preveniu os utilizadores de encontrarem erros e terem de travar o seu trabalho para resolver a situação com a empresa.

O *workflow*, comparando com os outros dois, é dos menos usados pelo cliente, no entanto é o segundo em que se aplicou mais esforço neste projeto. Isto pode justificar-se por ser o primeiro módulo trabalhado neste projeto, onde ainda se estava a identificar como se ia trabalhar e como funcionava o *software*, desta forma não foi planeado um tempo limite para este *script* para se poder investigar cada detalhe da aplicação com calma e ficar um *script* desenvolvido de base corretamente.

Dos testes efetuados, resultaram eventos com o propósito de reportar *bugs* que foram identificados durante a testagem. Entre os erros mais comuns verificados encontram-se:

- Uma funcionalidade não funcionar especificamente numa versão enquanto nas outras comporta-se corretamente;
- Um novo desenvolvimento dar resultado a um *bug* num módulo diferente, mas ligado ao do desenvolvimento;
- A correção de um erro trouxe ao de cima um erro de gravidade maior;
- Validações de campos não implementadas.

Como se pode inferir dos exemplos acima, a maior parte dos erros verificados resultaram de erros simples humanos, que transmitem ao cliente uma má imagem da empresa e frustração com o *software* por ele não funcionar como prometido no contrato. Estes tipos de erros costumam escapar aos programadores por estarem com o ponto de vista de programador e não de utilizador. Ao ver com os olhos de utilizador e testar o programa como deve ser usado e como não deve ser usado, para ver se a aplicação deteta os casos incorretos, os erros saltam à vista. É desta forma que os *testers* de *QA* ajudam os programadores a manterem uma qualidade de código perfeita para manter um cliente feliz.

Para executar os testes foram usados os métodos explicados na Secção 3. Cada um teve um nível de atenção e esforço desigual, observado através da Figura 5.1.

Figura 5.1 Escala de esforço envolvido em cada método

O plano de testes envolveu um grande foco na investigação da documentação do *software* e da sua *interface*, de forma que nenhum caso de teste faltasse, tendo sido gasto a maior parte do tempo neste

estudo. Contudo, passar a informação obtida para casos de teste escritos, não é uma tarefa rápida. Tem de se escrever todos os passos que o utilizador realiza, para que, qualquer pessoa que veja o documento consiga reproduzir os testes.

A parte de dados, vem logo a seguir ao plano de testes, não porque foi gasto muito tempo neste método, mas sim por causa da dificuldade de construir uma base de dados com todos os requisitos para executar os *scripts*. Muitas das vezes é preciso ir configurar detalhes de módulos diferentes do de teste, como têm funcionalidades interligadas, assim, é fácil de uma pessoa se perder no meio da configuração e acabar por ter uma base de dados que precisa de ser corrigida sempre que se corre um *script*.

Os *scripts* em si é são uma tarefa relativamente fácil, como no plano de testes já está descrito o que fazer e como. Pode haver alguns imprevistos na *interface* que não tenham sido documentados pelo plano de teste e, claro, erros de compilação, mas esses são rápidos de identificar e resolver. Mesmo a própria manutenção dos *scripts* é realizada de forma rápida, devido à boa documentação das alterações efetuadas ao código do programa.

Por último vêm os eventos, visto que o trabalho envolvido é estudar a informação do eventos, elaborar casos de teste, testar nas versões pedidas e escrever os resultados. Este processo pode sempre ser ajudado através de comunicação com o programador associado ao evento, o que torna este método o mais fácil e rápido.

Seguindo as discussões faladas anteriormente em conjunto com os resultados da Secção 4, é possível responder à pergunta de pesquisa deste projeto: Os testes e *scripts* aumentam a qualidade do código?

A resposta é um definitivo “sim”, basta olhar para o tipo de módulos com que se trabalhou neste projeto. Os módulos são todos de implementação recente então não tinham nenhum tipo de teste automático que testasse todas as suas funcionalidades, e, como foi exposto na Secção 2.1, os testes manuais não são capazes de detetar a maior parte dos *bugs* de uma aplicação, visto que não é realista poder testar todo o tipo de inputs de uma funcionalidade manualmente sempre que haja uma alteração no código.

Ao realizar os *scripts* foram detetados os erros que tinham escapado à equipa de programação, no caso do módulo de *workflows*, 42, no de assistências técnicas, 24, e no de produção, 32. Se estes *bugs* não tivessem sido detetados, iria-se continuar a desenvolver código de novas funcionalidades por cima deste o que no futuro, quando a junção dos *bugs* com as alterações fizesse com que os *bugs* fossem detetados, seria mais difícil de corrigir o código pois já teria as novidades da aplicação codificadas por cima, e esta tentativa de correção provavelmente iria dar origem a mais *bugs* nestes módulos. Com os erros detetados numa fase inicial dos módulos evitasse esta situação, o que aumenta a qualidade do código no momento e para o futuro, por estar sem erros e pronto para desenvolver novas funcionalidades.

6 Trabalhos relacionados

Com o desenvolvimento de técnicas de testagem de *software*, várias ferramentas capazes de testar diferentes tipos de programas foram realizadas com o objetivo de ajudar os *testers* de *software* na automatização do seu trabalho. No entanto, com várias ferramentas veem múltiplas escolhas de qual utilizar e quando. Assim, nos últimos anos, foram realizados estudos que comparam ferramentas de automatização de testes disponíveis no mercado a escolha de qual usar seja eficaz, mais rápida e certa. Nas próximas Subsecções são descritas três das mais populares ferramentas de executar testes automáticos disponíveis, as suas caraterísticas e, no final, qual a melhor a ser usada no meu projeto e porquê.

6.1 TestComplete

O *TestComplete* é uma ferramenta que suporta aplicações de *web*, *desktop* e *mobile*. Tem também suporte para as linguagens: *JavaScript; C++; Python; JScript; VBScript; DelphiScript ; C#.*

Uma das suas maiores caraterísticas é que tem um *UI* acessível e pronto para testar aplicações do ponto de vista dos seus utilizadores. Para tal, a pessoa que está a trabalhar com esta ferramenta precisa de ter conhecimentos de programação e conhecimentos para a ferramenta, que podem ser obtidos através da sua ampla documentação e tutoriais disponíveis no site da empresa que desenvolveu a ferramenta (SmartBear, 2022).

Por outro lado, só suporta realizar os testes no ambiente *Windows*, ou seja, no caso de se estar a testar um programa *mobile* é preciso abri-lo num *emulador*, uma aplicação que permite executar programas de *mobile* em *desktop*, e não tem nenhum mecanismo de segurança sobre as bases de dados que são utilizadas durante os testes (Gamido & Gamido, 2019).

6.2 Selenium IDE

A ferramenta *Selenium IDE* é um *addon* para o *browser Firefox* desenvolvido por Jason Huggins, como tal a ferramenta é usada na testagem de aplicações *web*. A ferramenta suporta as linguagens de programação: *Java*; *.Net*; *Perl*; *PHP*; *Python*; *Ruby*.

Um utilizador precisa de ter conhecimentos de programação para fazer uso desta ferramenta, no entanto a própria ferramenta é simples de utilizar devido ao facto de só suportar um tipo de aplicações. Por outro lado, a sua simplicidade é paga com a velocidade de testagem. Comparando a outras ferramentas do mercado, o Selenium IDE demora mais tempo na execução de cada caso de teste. (Kaur & Gupta, 2013).

6.3 Quick Test Professional

O *Quick Test Professional* é uma ferramenta de automatização de teste gráfica desenvolvida pela *HP*. Esta ferramenta é usada em casos em que a aplicação a testar precisa que sejam realizados testes funcionais, usados para avaliar a conformidade e os requisitos de um sistema, e testes de regressão, onde se garante que não surgiram novos defeitos em componentes que já foram anteriormente analisados. Como as outras ferramentas, o *Quick Test Professional* suporta várias linguagens: *VBScript; Java; .Net; Delphi*.

Para fazer uso desta ferramenta não é preciso ter conhecimentos profundos de programação, o básico dos básicos é suficiente para entender como a ferramenta funciona. Contudo, em vez de investimento em conhecimento, é preciso fazer um grande investimento de recursos para conseguir cobrir todos os casos de teste necessários (Gamido & Gamido, 2019).

6.4 Critério de seleção da ferramenta

Ao investigar cada uma das ferramentas referidas nas subsecções anteriores é possível concluir qual a melhor ferramenta a usar para o programa *ARTSOFT ERP*.

No caso do *Quick Test Professional*, visto que as pessoas na empresa têm experiência em programação, é necessário fazer grande investimento de recursos, o que torna esta ferramenta a menos adequada para o *ERP*.

Com o *Selenium IDE*, apesar de usar um maior tempo na execução dos testes, é uma ferramenta bastante simples de utilizar, o que a torna numa potencial concorrente a ferramenta a ser usada para os testes neste projeto. Contudo, o programa da *ARTSOFT* é uma aplicação para *desktop*, enquanto o *Selenium IDE* só consegue executar testes sobre aplicações de *web*.

Com isto, o *TestComplete*, que é uma opção adequada nesta situação, uma vez que suporta aplicações de *desktop* e também é uma ferramenta com um *UI* simples de compreender. Apesar de não ter segurança sobre as bases de dados que são usadas durante os testes, isto não compõe um problema. No caso dos testes realizados durante o projeto não é necessário haver segurança apertada, pois as bases de dados usadas são geradas ou manipuladas pelos *testers* de modo que não contenham dados reais dos clientes.

7 Conclusões

Com o trabalho desenvolvido neste projeto, foram realizados vários tipos de testes, automáticos e manuais. Para poder executar algum dos dois tive de ter intensa atenção a cada detalhe da informação sobre *software*, *ARTSOFT ERP*.

Foram elaborados três *scripts* para três módulos diferentes: *workflows*, assistências técnicas e produção. Os *scripts* foram escritos com a linguagem de programação *C#* com o objetivo de validar os dados que ficavam guardados nas bases de dados à medida que eram executadas operações. O trajeto da escrita e execução de *scripts*, apurou-me o sentido para a escrita de casos de teste, assim, à medida que o projeto se desenvolvia fui capaz de identificar, mais e mais *bugs*, com casos de teste cada vez mais detalhados.

O objetivo principal que me foi definido para este projeto foi de desenvolver *scripts* e testes manuais para poder testar vários módulos e funcionalidades que me forem encarregues, quando há uma nova versão do programa para ser lançada. Outro objetivo, que está inerente à escrita de *scripts*, é ter os *scripts* atualizados, ou seja, ter sempre os *scripts* bem documentados e explicados para, quando for necessário fazer manutenção aos mesmos, devido a uma alteração no código, esta consume os menos recursos possíveis.

Ambos os objetivos foram compridos não só na totalidade, como também com amplo tempo. O último *script* que elaborei, assistências técnicas, foi dado como finalizado em meados do mês de abril, o último mês antes de me dedicar exclusivamente à escrita da tese. Os *scripts* foram executados em todas as versões lançadas do *ARTSOFT ERP* durante a duração do meu projeto e a sua manutenção foi-me atribuída e realizada sempre que havia uma alteração planeada para um dos módulos.

Concluindo, durante este projeto desenvolvi três *scripts*, com os seus respetivos planos de teste, capazes de testar todas as funcionalidades dos módulos e todo o tipo de *input*, quer válido e inválido. Desta forma, foram identificados todos os *bugs* que o código continha durante a execução dos testes automáticos, o que contribuiu eficazmente para um *software* sem erros graves e uma melhor confiança entre a empresa e os seus clientes.

7.1 Ameaças

Apesar de ter sido um trabalho com resultados prometedores, não se pode deixar de pensar em possíveis ameaças que podem deitar abaixo todo o projeto desenvolvido. Uma vez que, lá por estar completamente funcional no momento, nunca se sabe o que vai ser alterado no futuro que faça com que o projeto não seja compatível com essas mudanças.

A maior ameaça ao meu projeto é a alteração de funcionalidades correntes *no ARTSOFT ERP*, isto por causa dos *scripts* escritos. Os *scripts*, uma vez a correr, estão à espera de encontrar os mesmos campos, botões, caixas, etc. que foram escritos no código quando se estavam a desenvolver os testes.

Assim, quando houver uma mudança, por exemplo, o botão que gerava um documento foi integrado noutro menu ou decidiram retirar um dos campos que dantes era para preencher com uma data, o *script* fica parado na sua execução pois está à procura do botão/campo que está indicado no código. Se passar um certo tempo sem encontrar, o *script* pára a execução, não corre o resto dos casos de teste e o resultado do teste é um erro.

É por esta razão que a manutenção de *scripts* é tão importante como o próprio desenvolvimento. Através da manutenção, sempre que haja alguma alteração no código do programa, altera-se, também o *script*, para ficarem os dois sempre atualizados um com o outro.

7.2 Trabalho Futuro

Caso fosse possível continuar o projeto ou caso alguém da equipa de *QA* da *ARTSOFT* fique encarregue dos *scripts*, as seguintes tarefas poderiam ser feitas:

- Manutenção dos *scripts* sempre que alguma funcionalidade seja mudada, mesmo que seja só o nome de um campo;
- Manutenção dos planos de teste à medida que o *software* vai crescendo, visto que vão sendo adicionadas novas funcionalidades aos módulos, conforme as necessidades dos clientes, e estas precisam de ser comtempladas nos *scripts* com os seus próprios casos de teste;
- Execução dos *scripts* sobre versões a serem lançadas e, caso um *bug* seja detetado, a criação de eventos a reportar a situação para que possam ser executados os *scripts* oura vez com a correção realizada para verificar se não existem mais *bugs* na versão, antes de chegar aos clientes;
- Realização de testes manuais sobre eventos que foram reportados por clientes, de forma a verificar pelo ponto de vista de utilizador, que os mesmos estão resolvidos;
- Elaboração de planos de testes de *scripts* para outros módulos do *ARTSOFT* que ainda não tenham *scripts* para testagem automática desenvolvidos, de modo que todos os módulos e todas as funcionalidades que o *ARTSOFT ERP* fornece sejam possíveis de ser testadas automaticamente, para poupar recursos e acelerar o processo de identificação e correção de erros.

8 Bibliografia

ARTSOFT. (19 de 04 de 2022). *ERP Software para grandes empresas - ARTSOFT - Software de Gestão*. Obtido de artsoft: https://www.artsoft.pt/erp-software/

Gamido, H., & Gamido, M. (2019). Comparative review of the features of automated software. *International Journal of Electrical and Computer Engineering (IJECE)*, 4473-4478.

javatpoint. (19 de 04 de 2022). *Automation Testing vs Sanity Testing - javatpoint*. Obtido de javatpoint: https://www.javatpoint.com/automation-testing

javatpoint. (19 de 04 de 2022). *Manual Testing - javatpoint*. Obtido de javatpoint: https://www.javatpoint.com/manual-testing

Kaur, H., & Gupta, G. (2013). Comparative Study of Automated Testing Tools: Selenium, Quick Test Professional and Testcomplete. *Int. Journal of Engineering Research and Applications*, 1739-1743.

Loyola-González, O. (2019). Black-Box vs. White-Box: Understanding Their Advantages and Weaknesses From a Practical Point of View. *IEEE Access*, 154096-154113.

Microsoft. (20 de 04 de 2022). *Noções básicas da base de dados*. Obtido de support.microsoft: https://support.microsoft.com/pt-pt/office/no%C3%A7%C3%B5es-b%C3%A1sicas-da-base-de-dados-a849ac16-07c7-4a31-9948-3c8c94a7c204?msclkid=879f121dc0c211eca319236e2f8c337b

SmartBear. (20 de 04 de 2022). *About File Checkpoints | TestComplete Documentation*. Obtido de support.smartbear: https://support.smartbear.com/testcomplete/docs/testing-with/checkpoints/files/about.html

SmartBear. (27 de 04 de 2022). *TestComplete 15 Documentation | TestComplete Documentation*. Obtido de support.smartbear: https://support.smartbear.com/testcomplete/docs/index.html

SmartBear. (20 de 04 de 2022). *TestComplete Automated UI Testing Tool | SmartBear*. Obtido de smartbear: https://smartbear.com/product/testcomplete/overview/

StackOverflow. (8 de Junho de 2022). *Stack Overflow Developer Survey 2021*. Obtido de stackoverflow.com: https://insights.stackoverflow.com/survey/2021#technology