Mestrado em Engenharia Informática
Dissertação/Estágio
Relatório Final

Banca Internacional – CRITICAL Software

João Francisco Cabral de Carvalho Almeida
jfcabral@student.dei.uc.pt

Orientador (DEI-FCTUC):
Prof. Doutor Marco Vieira

Orientador (Critical Software):
Eng. Telmo Simões

1 de Setembro de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Mestrado em Engenharia Informática
Dissertação/Estágio
Relatório Final

Banca Internacional – CRITICAL Software

João Francisco Cabral de Carvalho Almeida
jfcabral@student.dei.uc.pt

Júris:

Prof. Doutor Carlos Fonseca

Prof. Doutor César Teixeira

1 de Setembro de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

Num mundo cada vez mais exigente e competitivo, os sistemas de informação assumem um papel fulcral no dia-a-dia das organizações, contribuindo para a sustentabilidade e manutenção do seu negócio. Não obstante, o crescimento e desenvolvimento de uma empresa depende da sua capacidade de inovar e de se re-inventar, garantindo um melhor serviço aos seus clientes. De forma a acompanhar esta tendência de melhoria contínua, os sistemas de informação necessitam de evoluir a par com as organizações, contribuindo para um aumento da eficiência, redução de custos e consequentemente maior competitividade.

O principal objetivo deste projeto de estágio consiste na integração do estagiário na equipa do projeto *eMudar* entre a *Critical Software* e o Banco de Fomento de Angola, com vista a promover o seu desenvolvimento a nível pessoal e profissional através do exercício daquelas que são as principais atividades de engenharia da empresa: Análise Funcional, Desenvolvimento e Garantia de Qualidade e Testes.

O projeto *eMudar* visa o desenvolvimento e a manutenção evolutiva de um sistema de informação, feito por medida, que suporta as principais tarefas efetuadas em *backoffice* nos diversos órgãos que constituem o banco. Das suas múltiplas valências, destaca-se a definição, melhoria e uniformização de processos de negócio, bem como a sua desmaterialização, garantindo uma gestão integrada de documentos em formato digital.

Enquanto Sistema de Informação de caráter crítico, é necessário assegurar um elevado nível de qualidade, o que é garantido através da aplicação das boas práticas da Engenharia de Software preconizadas pela empresa.

O contributo do estagiário contempla assim a participação em dois módulos distintos do sistema: o Módulo de Relatórios e o Módulo de Organização e Autorização.

Palavras-Chave

Sistema de apoio à atividade bancária, Sistema *Business-Critical*, Integração de Sistemas, Sistemas de Informação

Abstract

In an increasingly demanding and competitive world, information systems assume a central role in every organization's daily reality, and contribute to the sustainability and management of their enterprise. Notwithstanding, the growth and expansion of a company depends on its ability to innovate and reinvent itself, and provide the best possible service to its clients. In order to follow this constant improvement trend, information systems need to progress along with organizations, adding efficiency increase, reduction in costs, and eventually greater competitiveness to them.

The main goal of this internship project consists of integrating the trainee into the *eMudar* team, working with Critical Software and Banco de Fomento Angola, in order to promote his own personal and professional development, by ensuring the participation in the most significant engineering activities of the company: Functional Analysis, Development and Quality Assurance.

eMudar Project is designed to develop and ensure the evolutionary maintenance of a customized information system, which is, and supports the main tasks performed in backoffice by the different sections of the bank. Among its multiple attributes are the specification, improvement and standardization of business procedures as well as their dematerialization, providing an integrated management of documents in a digital format.

As a critical Information System, there is the need to ensure high quality standards, which is guaranteed by means of applying the good Software Engineering practices established within the company.

The role of the trainee includes the participation in two distinct modules of the system: a Reporting Module, and the Authorization and Access Control Module.

Keywords

Banking Support System, Business-Critical System, Enterprise Systems Integration, Information Systems

Para ser grande, sê inteiro: nada
Teu exagera ou exclui.
Sê todo em cada coisa. Põe quanto és
No mínimo que fazes.
Assim em cada lago a lua toda
Brilha, porque alta vive

— Ricardo Reis, Odes –

Fernando Pessoa

Índice

Lista de Figuras

Lista de Tabelas

Glossário

API *Application programming interface*

BANKA Designação do sistema *core bancário* legado, assente em tecnologia IBM onde são realizadas a generalidade das transações bancárias.

BFA Banco de Fomento Angola

BPM *Business Process Management*, É uma abordagem de gestão que promove a eficácia e a eficiência do negócio, ao mesmo tempo que preconiza a inovação, a flexibilidade e a melhoria contínua[2] através da modelação e otimização dos processos de negócio.

BPMS *Business Process Management Suite*, Conjunto de ferramentas de suporte que automatizam o desenho, modelação, execução, simulação, integração, monitorização e otimização de processos de negócio[2].

CMMI *Capability Maturity Model*

CR *Change-Request*

CRM *Customer Relationship Management*, Conjunto de estratégias e técnicas com vista a gerir e monitorizar as interações com clientes, garantindo um acompanhamento individualizado, visando a melhoria da qualidade dos serviços prestados[3].

DDL *Data Definition Language*, Linguagem de definição de dados - linguagem padronizada utilizada para a definição de estruturas de dados em bases de dados.

DEI *Departamento de Engenharia Informática*

GUI *Graphical User Interface*, Interface Gráfica do Utilizador

JIRA Ferramenta de suporte à gestão e acompanhamento de projetos, bem como à monitorização de defeitos. Desenvolvido pela Atlassian, o JIRA apresenta uma arquitetura modular, podendo ser facilmente extensível através de *plugins*, sendo portanto bastante flexível.

JSON *JavaScript Object Notation*

JVM *Java Virtual Machine*

MVC *Model–view–controller*, Padrão arquitetural vocacionado para a criação de interfaces de utilizador, promovendo uma separação clara entre a lógica de visualização e a lógica de negócio, resultando num menor acoplamento e consequentemente maior modificabilidade[4][5].

MVP *Model–view–presenter*, Padrão arquitetural variante do *MVC*, onde a camada *Presenter* media a comunicação entre o *Model* e a *View*, assumindo também a função de controlador[5].

NPAPI *Netscape Plugin Application Programming Interface*

RDBMS *Relational Database Mnagement System*

REST *Representational State Transfer*

SaaS *Software as a Service*, Software como serviço

SDLC *Software Development Life Cycle*

SOA *Service-Oriented Architecture*, Estilo arquitetural que assenta na criação de serviços interoperáveis, de baixo acoplamento entre si, que podem ser facilmente reutilizáveis e integráveis noutros contextos[6].

SOAP *Simple Object Access Protocol*

SPAE *Software Product Assurance Engineer*, Colaboradores responsáveis pela garantia de qualidade de *software*, participando ativamente na melhoria de processos, especificação e execução de testes de software, triagem de defeitos, recolha e análise de métricas de qualidade e até revisão de requisitos. São tipicamente dos membros de um projeto com mais conhecimento acerca do seu contexto, objetivos e âmbito.

SQL *Structured Query Language*

SUT *System Under Test*, Sistema em teste.

TCA *Total Cost of Aquisition*, Custo Total de Aquisição de bens ou serviços.

TTM *Time-to-Market*, Tempo de colocação de um produto no mercado.

UML *Unified Modeling Language*

WBS *Work Breakdown Structure*, Uma Estrutura Analítica de Projeto é um processo que consiste na decomposição hierárquica das tarefas que constituem um projeto, fornecendo uma visão estruturada e consequentemente um maior entendimento acerca da complexidade e abrangência do mesmo.

WSDL *Web Services Description Language*

XML *eXtensible Markup Language*[7]

ZK *Framework* para auxiliar o desenvolvimento da camada gráfica em aplicações *Web*. Escrita em Java e disponibilizada em regime de *open-source*, oferece um elevado conjunto de componentes extensíveis e reutilizáveis,

permitindo desenvolver interfaces ricas, abstraindo a utilização de *Javascript* e *Ajax*.

Capítulo 1

Introdução

O século XXI tem sido fortemente marcado pela terceira revolução industrial. No dia-a-dia de uma sociedade moderna a tecnologia tem vindo a assumir um papel cada vez mais preponderante, influenciando a forma como as pessoas vivem e interagem entre si. Associada a esta evolução, também as organizações e a forma como conduzem o seu negócio têm vindo a sofrer profundas e marcantes alterações, constituindo hoje um novo paradigma denominado a "Era da Informação"[8].

Atualmente, a informação assume-se como um recurso cada vez mais relevante e valioso, contribuindo para que as organizações se conheçam melhor a si próprias, a par dos seus clientes e do mercado onde operam. Os sistemas de informação mostram-se cada vez mais versáteis e capazes de auxiliar não só, nas tarefas do dia-a-dia, mas também na gestão, planeamento e tomada de decisão. Assim, independentemente da dimensão e do setor das organizações, estes sistemas tornam-se fulcrais para o desenvolvimento e manutenção do negócio, auxiliando-as a fazer face a um mundo cada vez mais globalizado, dinâmico e exigente.

1.1 Enquadramento

A **entidade acolhedora** encontra-se dividida em duas partes, a entidade gestora do estágio e a entidade a qual pertence o projeto. A primeira, a iTGrow[1] constitui um A.C.E. (*Agrupamento Complementar de Empresas*) com a participação equitativa da Critical Software e do BPI[2]. Tem como atividade principal o recrutamento e desenvolvimento de relações de parceria com escolas de engenharia, privilegiando a celebração de protocolos de estágio curricular e profissional. Tem ainda como objetivo a formação e qualificação dos seus empregados, de forma a prepará-los para a vida profissional. A segunda, a entidade promotora do projeto - a Critical Software, é uma empresa criada em 1998 por três alunos de doutoramento (à época) da Universidade de Coimbra, na área de sistemas críticos, através do *csXception*, um ambiente de injeção de falhas automatizado para avaliar e validar sistemas críticos, sendo considerada durante vários anos a área com maior enfoque da empresa. Atualmente, a empresa tem privilegiado a sua ação ligada aos sistemas críticos, estando presente em diversos tipos de mercados, a saber, a Defesa, Energia, Transportes, Comunicações, Aeronáutica, Espaço, Finanças, Saúde e Estado / Setor Público. Tem privilegiado um conjunto de parceiros altamente conceituados, tais como: a NATO[3], a ESA[4] e a NASA[5]. Detém diversas certificações[6], destacando-se *Capability Maturity Model Integration* [9][10] (CMMI) nível 5, ISO 9001, ISO 9100, ISO 15504, TickIT, SPICE e AQAP NATO. A nível de expansão do negócio, a empresa encontra-se atualmente em países como Portugal, Estados Unidos da América, Inglaterra, Alemanha, Ucrânia, Brasil, Angola e Moçambique.

Fundado em 1993, o **Banco de Fomento Angola** é a maior instituição bancária privada angolana, contando com mais de 1.4 milhões de clientes e 188 balcões[11]. Integrava à data 50.1% de participação pelo banco BPI e 49.9% pela Unitel[12].

[1]Página oficial da iTGrow: `http://www.itgrow.pt`

[2]Página oficial do banco BPI: `http://www.bancobpi.pt`

[3]Página oficial da NATO: `http://www.nato.int`

[4]Página oficial da ESA: `http://www.esa.int`

[5]Página oficial da NASA: `https://www.nasa.gov`

[6]Certificações de qualidade da *Critical Software*: `http://www.criticalsoftware.com/pt/how-we-do-it/quality`

Hoje, a organização encontra-se fortemente empenhada na melhoria da satisfação das necessidades da população Angolana, focalizando a sua ação na modernização do sistema financeiro do país e constituindo-se como líder na presença de Cartões de Débito e de Terminais de Pagamento Automático[11].

1.2 Projeto *eMudar*

De forma a manter uma posição líder no mercado, o BFA comprometeu-se em modernizar o seu sistema de informação interno para fazer face às necessidades do mesmo, visando o aumento da eficiência e da qualidade dos serviços prestados. Assim, surge em 2011 o **projeto *eMudar*** com vista à criação de um Sistema de Informação feito à medida, integrando a *infraestrutura legada* denominada *BANKA*, o que levou à uniformização dos múltiplos serviços disponibilizados nos balcões em *backoffice*.

Tal como qualquer outro sistema de informação, foi essencial começar por rever e otimizar os processos de negócio[13][14], uniformizando-os transversalmente ao longo de toda a organização, proporcionando a melhoria da eficiência com uma consequente redução de custos[15].

Este novo sistema procura colmatar as principais limitações da *infraestrutura legada*, a saber:

- Inadequação do sistema de autorizações, dificultando um eficaz controlo operacional;
- Complexidade da interface de utilização e dispersão da informação por muitos ecrãs/percursos, contribuindo para uma elevada curva de aprendizagem;
- Inexistência de suporte à automatização, controlo e monitorização de processos para processos bancários com múltiplos intervenientes;
- Complexidade no registo de algumas operações que obrigam a múltiplas ações fragmentadas e dependentes de processos manuais, resultando num esforço significativo por parte dos serviços centrais;

- Ausência de um sistema de gestão documental de suporte aos processos bancários.

Com base nos requisitos transmitidos pelo BFA, e através da experiência da empresa em projetos semelhantes, foram identificados ainda os seguintes objetivos complementares para o sistema[16]:

- Redução de riscos operacionais através da implementação de uma hierarquia de controlo de acesso com alertas;
- Automatização de processos manuais baseados em circulação de papel;
- Adoção progressiva de um modelo orientado por processos em oposição a um modelo baseado em transações;
- Formalização dos processos de negócio em formato *standard*, promovendo uma maior simbiose entre o sistema e o negócio;
- Captura de regras de negócio e normas internas de tarefas manuais, transpondo-as para o sistema;
- Automatização da atribuição de tarefas e respetiva monitorização;
- Redução de potenciais erros de inserção de dados que implicariam intervenções corretivas por parte dos serviços centrais;
- Disponibilização de uma visão integrada e consistente da informação;
- Inclusão de funcionalidades de CRM no *Front-Office* de forma a melhorar a eficácia e eficiência da atividade comercial.

O sistema assenta em tecnologias *web* e foi idealizado segundo uma *Arquitetura Orientada a Serviços* (SOA), facilitando a integração de sistemas externos existentes atualmente e salvaguardando a interoperabilidade com outros no futuro.

1.3 Contexto

O presente estágio prevê a participação em dois módulos distintos do sistema *eMudar*: o Módulo de Relatórios e o Módulo de Organização e Autorização, que são descritos nas sub-secções seguintes.

1.3.1 Módulo de Relatórios

Atualmente foi identificada a necessidade de gerar e apresentar um conjunto de relatórios a diferentes membros dos órgãos de gestão de médio e de alto nível. No sentido de colmatar esta situação, procedeu-se à utilização de um mecanismo informal de envio de relatórios simples no corpo de uma mensagem de correio eletrónico. Embora eficaz, esta solução apresentou várias limitações, das quais destacamos as seguintes:

Persistência dos dados - Os dados que constituem um relatório não são armazenados em nenhum local, sendo perdidos logo após o seu envio o que implica uma ausência de histórico e de suporte a alterações de visualização;

Visualização - Ainda que seja possível incluir *HTML* no corpo de um email, não é possível criar uma interface tão completa como numa aplicação nativa ou página *web*;

Relevância Temporal - Esta estrutura torna-se igualmente desadequada para a exibição de dados em tempo real;

Integração - Dado o seu carácter estático, não é exequível a interação com o sistema *eMudar*, nomeadamente ao nível do desencadeamento de ações através do próprio relatório;

Extensibilidade - Não pode ser facilmente estendida ou modificada.

Para colmatar as principais limitações com esta abordagem, pretende-se criar um módulo que facilite a visualização de relatórios de forma integrada no sistema *eMudar*. Destaca-se, no entanto, que os requisitos deste módulo já se encontram

elaborados pela equipa de Análise Funcional do projeto, tendo sido aprovados pelo cliente, ficando assim esta tarefa fora do âmbito do estágio.

1.3.2 Módulo de Organização e Autorização

O Módulo de Organização e Autorização é uma das componentes mais importantes e críticas do sistema, sendo responsável pela manutenção de uma estrutura hierárquica de permissões, assegurando o controlo de acesso à plataforma. Este controlo é feito através de perfis que determinam o acesso a certo tipo de funcionalidades, sendo responsabilidade da gerência a atribuição de perfis atendendo às competências e responsabilidades de cada utilizador.

Apesar de já se encontrar em produção, este módulo necessita de alterações periódicas de forma a refletir tanto mudanças na estrutura interna da organização, como também em outros aspetos relativos ao negócio e aos respetivos processos.

1.4 Objetivos do Estágio

Este estágio curricular prevê a participação nas três principais atividades de um Engenheiro de Software:

Análise Funcional
Esta componente, efetuada no âmbito do módulo de Organização e Autorização, envolve as seguintes tarefas:

- Análise do módulo e dos pedidos de alteração por parte do cliente;
- Prototipagem de média fidelidade e elaboração de um *storyboard*, possibilitando a realização de um *software walkthrough*;
- Reformulação e atualização da especificação de requisitos existente;
- Compilação de um resumo das alterações a serem implementadas.

Implementação
Esta componente, realizada no contexto do Módulo de Relatórios, engloba a seguintes as tarefas:

- Análise Técnica (processo definido na secção 3.2.5);
- Desenho da arquitetura da componente;
- Desenho do esquema de base de dados;
- Codificação do módulo.

Garantia de Qualidade e Testes
Finalmente, a participação neste domínio foi feita em relação ao Módulo de Relatórios, envolvendo as seguintes tarefas:

- Inspeção e revisão dos requisitos do módulo;
- Elaboração de um Plano de Testes;
- Definição de Testes de Aceitação;
- Execução dos Testes e documentação dos defeitos encontrados.

Para além destas tarefas, foram ainda identificados alguns objetivos adicionais relevantes que, caso sejam atingidos, representam uma mais-valia para o estágio e para o projeto:

Automatização de Testes
Dado o caráter crítico do projeto, a fase de testes é fundamental para encontrar com a maior antecedência eventuais defeitos, minimizando-se assim as consequências e os custos associados[17]. Não obstante, dada a dimensão e complexidade do projeto, os testes de aceitação são feitos de forma manual, o que requer um elevado esforço humano antes de cada *release*. Assim, seria desejável que os testes especificados e realizados sob a ótica de *SPAE* incluíssem a sua automação, como forma de minimizar o esforço despendido em testes a médio prazo. Neste sentido,

destaca-se que a concretização desta tarefa constitui um protótipo para a automatização parcial de uma etapa crucial do *SDLC*, podendo contribuir para a adoção de práticas modernas de engenharia, acrescentando valor às mesmas.

Testes de Robustez

No âmbito da qualidade de software, é frequente concentrarem-se os esforços na verificação do comportamento do sistema face ao que é esperado, de acordo com os requisitos[18]. No entanto, considera-se que, na ausência de testes de robustez, se torna difícil de prever qual o comportamento do mesmo face a situações inusitadas, não previstas nos requisitos, e a cenários extremos. Dado o contexto do projeto, considera-se que este objetivo seria vantajoso, contribuindo para a melhoria da qualidade do produto.

Participação em *CRs* adicionais

Entende-se por *CR* todo e qualquer pedido de alteração sobre um plano ou produto que tenha sido previamente aceite, podendo ser originado por *stakeholders* de origem interna ou externa. Pode assumir várias formas, destacando-se as alterações de *Âmbito* (adição, modificação ou remoção de funcionalidades), de *Calendário* e de ordem *Tecnológica*. De forma a alcançar um melhor entendimento acerca do funcionamento do sistema, a ação do estagiário pode passar pela participação em *CRs* adicionais, estendendo a sua intervenção no projeto e contribuindo para um estágio mais completo e abrangente.

Para a concretização destes objetivos, pressupõe-se a integração do estagiário na equipa do projeto *eMudar*, em ambiente real, dotando-o de conhecimentos concretos sobre processos e boas práticas de desenvolvimento de sistemas críticos, concretamente na área da banca. Esta estratégia de integração em contexto real revela-se não só, como uma vantagem inclusiva do estagiário no ambiente de trabalho, mas também com uma formação profissional adaptada ao mercado de trabalho e às exigências cada vez mais desafiadoras do desenvolvimento de *software*.

1.5 Condicionantes

Existem algumas restrições que, no nosso entender, afetaram o desempenho do trabalho do estagiário, podendo ser agrupadas de acordo com o seu carácter, em quatro domínios:

Temporal

Durante o primeiro semestre deste estágio curricular, o contributo por parte do estagiário é limitado a 40%, dado que o Mestrado em Engenharia Informática pela Universidade de Coimbra prevê que os discentes frequentem três unidades curriculares em simultâneo com as atividades de estágio. Salienta-se que esta limitação causa uma grande mudança de contexto(s) entre os compromissos académicos e os profissionais, afetando o desempenho do mesmo.

Negócio

Inserido num contexto de real de trabalho e *business-critical*, os níveis de exigência e de responsabilidade são naturalmente elevados pelo que, por vezes, os *timmings* de concretização do projeto poderão estar desfasados face aos objetivos do estágio, sobretudo, no que diz respeito ao nível de prioridades estabelecidas pelo cliente e disponibilidade de recursos humanos.

Tecnológico

Dado o tipo e a complexidade do projeto, torna-se fulcral utilizar tecnologias estáveis e amplamente testadas e confiáveis, em detrimento dos últimos avanços tecnológicos. No entanto, é importante enfatizar que este critério de seleção está associado a um desenvolvimento menos ágil e mais verboso, tornando-o moroso e pesado, refletindo-se nas estimativas efetuadas. Por outro lado, na presença de um projeto que já encontra em curso, torna-se necessário fazer uso das tecnologias sobre as quais o projeto assenta.

Geográfico

Na presença de um cliente angolano, a distância revela ser um fator condicionador, dado que afeta a frequência e qualidade da comunicação e ligação ao mesmo. De realçar a importância deste último fator, uma vez que o sistema *eMudar* está construído sobre a *infraestrutura legada* do banco (o sistema

BANKA), o que obriga a uma ligação permanente a Luanda. Qualquer disrupção nesta ligação afeta negativamente o desenvolvimento e impossibilita o processo de testes.

Pessoal

Neste ponto destaca-se a inexperiência do estagiário no processo de desenvolvimento de sistemas críticos, bem como na área de garantia de qualidade e testes, o que obriga a um dispêndio de tempo essencial em formação e treino, antes do início das tarefas propostas.

1.6 Estrutura do Documento

O presente relatório encontra-se dividido em sete capítulos:

Neste 1º capítulo apresentamos o enquadramento do estágio, onde é caracterizada a entidade acolhedora, o cliente e o contexto do projeto *eMudar*. De seguida são descritos os objetivos do estágio e adas as principais condicionantes identificadas.

No 2º capítulo efetuamos uma análise comparativa das tecnologias com maior relevância para o trabalho desenvolvido no estágio, nomeadamente: Servidores de Aplicação de *Java EE* e ferramentas para a automatização de Testes.

No 3º capítulo são descritos os principais processos de engenharia que foram instanciados no decorrer do estágio, assim como o planeamento das atividades para os dois semestres que integram o mesmo.

No 4º capítulo aborda-se o desenvolvimento do Módulo de Relatórios, contemplando a sua arquitetura, o seu modelo de dados e a descrição das tarefas inerentes a esta etapa.

No 5º capítulo apresenta-se todo o trabalho efetuado no âmbito da Qualidade de Software relativamente ao Módulo de Relatórios, destacando-se a automatização de testes de aceitação e os testes de robustez.

No 6º capítulo é apresentado o contributo a nível de Análise Funcional através da expansão e reformulação de uma parte dos requisitos do Módulo de Organização e Autorização.

Finalmente, no 7º capítulo, encontram-se as consideração finais e conclusão do relatório de estágio.

Capítulo 2

Tecnologias

Em qualquer projeto de Engenharia de Software é fundamental que a escolha das tecnologias a serem utilizadas seja feita de forma consciente, procurando endereçar as particularidades do problema em causa, contribuindo para a satisfação das necessidades do cliente.

Nesta secção, é feita uma análise sobre o *Java Enterprise Edition*, a principal tecnologia sobre a qual assenta o projeto *eMudar*, sendo comparados diversos servidores de aplicação existentes no mercado. Também foi feito um estudo sobre ferramentas para a automatização de testes de aceitação, contribuindo para o desenvolvimento das temáticas abordadas no Capítulo 5.

2.1 Java EE

O *Java Enterprise Edition* consiste numa especificação padronizada que tira partido da linguagem de programação Java, permitindo desenvolver uma plataforma orientada para a criação de aplicações sobretudo em ambiente empresarial. Na tabela 2.1 podem ser observados os atributos de qualidade definidos pela *Sun* durante a conceção desta especificação[1].

Apesar de muitas das suas vantagens estarem associadas ao próprio *Java* e à forte comunidade que existe em volta da linguagem, o Java EE define um conjunto

Tabela 2.1: Atributos de Qualidade do Java EE definidos pela *Sun*[1]

Quality Attribute	Requirement
Portability	J2EE should be able to be implemented with minimal work on a variety of computing platforms
Buildability	Application developers should be provided with facilities to manage, common services such as transactions, name services, and security
Balanced Specificity	Detailed enough to provide meaningful standard for component developers, vendors, and integrators, but general enough to allow vendor-specific features and optimizations
Implementation Transparency	Provide complete transparency of implementation details so that client programs can be independent of object implementation details (server-side component location, operating system, vendor, etc.)
Interoperability	Support interoperation of server-side components implemented on different vendor implementations; allow bridges for interoperability of the J2EE platform to other technologies such as CORBA and Microsoft component technology
Evolvability	Allow developers to incrementally adopt different technologies
Extensibility	Allow incorporation of relevant new technologies as they are developed

de ferramentas que facilitam a criação de *web-services*, a interação com bases de dados e filas de mensagens, o mapeamento objeto-relacional, a criação de páginas dinâmicas para a *web*, entre outros.

A sua decomposição em módulos pode ser observada na figura 2.1, onde se destacam a azul os componentes que foram introduzidos na sétima versão da especificação. Os módulos listados horizontalmente constituem uma base sólida para a criação de aplicações, permitindo abstrair alguns pontos sensíveis e delicados durante implementação de um sistema, contribuindo assim para um desenvolvimento mais agilizado. Já os módulos apresentados na vertical representam componentes de carácter transversal que permitem agilizar tópicos como: concorrência, injeção

de dependências e validação de parâmetros, tornando a especificação mais poderosa e flexível.

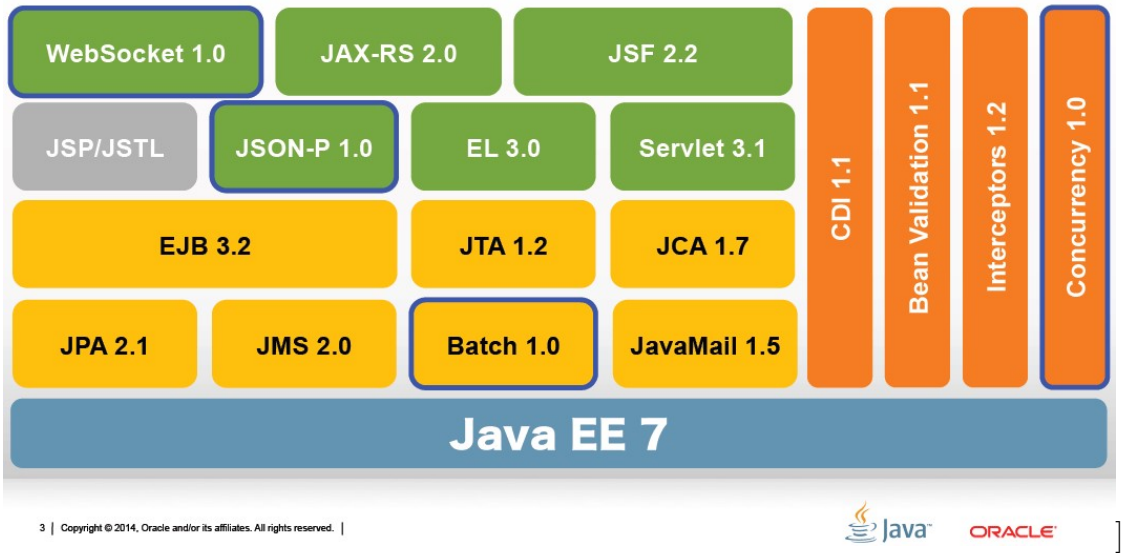


Figura 2.1: Representação dos diversos componentes disponíveis no Java EE 7[2]

Apesar do *Java EE* ser constituído por uma especificação única, existem várias implementações de referência, o que leva à necessidade da seleção de um servidor de aplicação, resultando na ausência de *vendor lock-in*[3]. Na secção seguinte apresentam-se aquelas que são as principais soluções de mercado, sendo alvo de uma comparação de carácter incremental.

[2]Origem da imagem: https://slideshare.net/OracleMiddleJP/java-ee-7-45-features
[3]https://sourcemaking.com/antipatterns/vendor-lock-in

2.2 Servidor de Aplicação de *Java EE*

São muitos os fatores que podem ser considerados na escolha de um servidor de aplicação; no entanto, antes de se avançar para um cenário de comparação, é importante compreender o contexto do cliente, bem como as suas necessidades e implicações das mesmas.

Tabela 2.2: Impacto Financeiro do tempo de inatividade por sector[4]

Industry Segment	**Cost per hour (Millions)**
Energy	$ 2.8
Telecommunications	$ 2.1
Manufacturing	$ 1.6
Financial	$ 1.5
Information Technology	$ 1.4
Insurance	$ 1.2
Retail	$ 1.1
Pharmaceuticals	$ 1.1
Banking	$ 1.0
Consumer Products	$ 0.8
Chemicals	$ 0.7
Transportation	$ 0.7

Atendendo ao setor de atividade em que o cliente se encontra inserido, a tabela 2.2 ajuda a reforçar a necessidade de desenvolver um sistema *business-critical*, promovendo a utilização de tecnologias amplamente utilizadas/testadas, estáveis e robustas, em detrimento das últimas tendências tecnológicas.

Na primeira etapa desta comparação pretende-se dar a conhecer aqueles que são os principais fornecedores de Servidores de Aplicação no universo do *Java EE*. As múltiplas ofertas são apresentadas em seguida, de acordo com quatro critérios base com vista a distinguir, mais facilmente, o tipo e o público alvo das diversas

[4]Origem dos dados: ftp://public.dhe.ibm.com/software/zseries/pdf/florida/handouts/6_Ray_Jones_System_z_Spring_Premier_Event_v2.pdf#page=20

soluções, a saber: Disponibilidade do Código, Licenciamento, Suporte Comercial e Versão de Conformidade do Java EE. De salientar que foram apenas consideradas implementações do *Java EE Full Profile* certificadas pela *Oracle* que implementam, no mínimo, a versão 6, datada de Dezembro de 2009[5].

Tabela 2.3: Servidores de Aplicação de *Java EE* certificados pela *Oracle*

Organização	**Produto**	**Código Aberto**	**Licença**	**Suporte comercial**	**Certificação Java EE**[a]
Apache	Geronimo	sim	livre	sim[b]	6
Hitachi	Cosminexus	não	comercial	sim	7
IBM	WebSphere	não	comercial	sim	7
IBM	WebSphere Community Edition	baseado	livre	não[c]	6
Oracle	GlassFish	sim	livre	não[d]	7
Oracle	WebLogic	não	comercial	sim	7
Red Hat	JBoss Enterprise Application Platform	sim	comercial	sim[19]	6
Red Hat	Wildfly	sim	livre	não[19]	7
TmaxSoft	Jeus	não	comercial	sim	7

[a] Certificação de conformidade com *Java EE Full Platform* pela Oracle[20]

[b] Apenas disponibilizado por terceiros[21][22]

[c] Fim do suporte anunciado para Setembro de 2016[22]

[d] Java EE and GlassFish Server Roadmap: `https://goo.gl/IoRoyk`

Observa-se na tabela 2.3 que os três mais antigos e mais prestigiados concorrentes no setor do *Middleware* presentes nesta lista: a *IBM*, a *Oracle* e a *Red Hat*, apostam em duas vertentes distintas de servidores de aplicação. Uma versão com foco comercial destinada a clientes empresariais, e uma versão comunitária disponibilizada gratuitamente em regime de código aberto, com um foco *bleeding edge*, tornando-se potencialmente mais instável, ficando portanto ausente de suporte comercial. Esta prática promove uma maior popularidade ao atrair a comunidade

[5]JSR 316: JavaTM Platform, Enterprise Edition 6 Specification - `https://jcp.org/en/jsr/detail?id=316`

open-source, favorecendo o desenvolvimento do seu ecossistema, contribuindo para a adoção das suas soluções comerciais.

De seguida, listam-se apenas soluções que incluem suporte comercial garantido pela própria organização, considerado como um dos requisitos mais importantes para este tipo de clientes.

Tabela 2.4: Comparação da quota de mercado de Servidores de Aplicação de *Java EE* comerciais

Organização	**Produto**	**Marketshare Gartner**[a]	**Marketshare Zeroturnaround**[b]	**Certificação Java EE**[c]
Hitachi	Cosminexus	n/a	<1%	7
IBM	WebSphere	29%	4%	7
Oracle	WebLogic	34%	8%	7
Red Hat	JBoss EAP	3%	16%[d]	6
TmaxSoft	Jeus	n/a	<1%	7

[a] De acordo os dados disponíveis[23] do levantamento efetuado pela *Gartner* durante o ano 2014[24]

[b] De acordo com um relatório patrocinado pela empresa *Zeroturnaround*, correspondente ao ano de 2014[25]

[c] Certificação de conformidade com *Java EE Full Platform* pela Oracle[20]

[d] Incluí a versão comunitária da *Red Hat*, o Wildfly

Na tabela 2.4 compara-se a presença de mercado das soluções comerciais em estudo. Para o efeito selecionaram-se duas métricas baseadas em estudos de mercado, um da *Gartner*, uma empresa de consultoria em TI americana e outro da *ZeroTurnaround*, uma *software house* oriunda da Estónia, conhecida pelo desenvolvimento do *JRebel*, um *plugin* de produtividade amplamente utilizado no contexto do *Java EE*. De salientar que os referidos estudos incluem servidores de aplicação que não implementam o *full profile* e consideram igualmente soluções comerciais e soluções livres.

Através dos dados disponíveis, observa-se que empresas como a *Hitachi* e a *TmaxSoft* apresentam uma reduzida percentagem de mercado, o que pode ser explicado dado o seu foco no mercado oriental, nomeadamente Japão e Coreia do Sul[26] respetivamente.

Seguidamente, comparam-se as alternativas restantes, de acordo com o seu desempenho num *benchmark standard* para servidores de aplicação de Java EE, o *SPECjEnterprise 2010*[6]. Nesta suite, a métrica de desempenho obtida é o *Enterprise jAppServer Operations Per Second* (EjOPS). No entanto, em virtude do *hardware* utilizado por cada organização não ser exatamente igual, utilizar-se-á como métrica de comparação o número de *EjOPS* por núcleo do processador. Destaca-se ainda que a *Red Hat* nunca submeteu resultados do seu Servidor de Aplicação, o que impossibilita a sua comparação através deste critério.

Tabela 2.5: Comparação do desempenho de Servidores de Aplicação através do *benchmark SPECjEnterprise2010*[6]

Organização	**Produto**	**EjOPs / Core**[a]	**EjOPs / Core**[b]
IBM	WebSphere	689[c]	939[d]
Oracle	WebLogic	522[e]	784[f]
Red Hat	JBoss EAP	n/a	n/a

[a] Em arquitetura *Intel* de 22nm
[b] Em arquiteturas proprietárias (*IBM Power8* e *Oracle SPARC M7* respetivamente)
[c] Execução submetida em Janeiro de 2015: `https://goo.gl/hbqdoG`
[d] Execução submetida em Março de 2014: `https://goo.gl/aT21ju`
[e] Execução submetida em Dezembro de 2014: `https://goo.gl/Cy5Rvu`
[f] Execução submetida em Outubro de 2015: `https://goo.gl/DhYvUo`

Observa-se na tabela 2.5 que a solução da *IBM* apresenta um desempenho por núcleo de cerca de 32% superior em arquitetura *Intel* e cerca de 20% superior com a sua arquitetura proprietária face à *Oracle*.

A nível de custo é importante salientar que as licenças deste tipo de soluções são muito diferentes entre si, dependendo muito do contexto em que irão ser utilizados[27][28]. Por este motivo, não sendo este um dos critérios mais significativos atendendo ao tipo de cliente em questão, optou-se por não o considerar para efeitos de comparação.

Relembrando a tabela 2.2 e atendendo ao setor de atividade a que o cliente se

[6]Página oficial do *SPECjEnterprise 2010*: `https://www.spec.org/jEnterprise2010`

encontra ligado, é necessário apostar numa solução estável e robusta. Assim, refletindo sobre os dados e métricas recolhidos, considera-se que as soluções comerciais da *IBM* e da *Oracle* são as mais adequadas para o projeto.

Não obstante, dado que a *infraestrutura legada* do banco está assente sobre tecnologia da *IBM*, e na ausência de motivos que justifiquem a migração para outro fornecedor, optou-se pela solução desta empresa, contribuindo para uma interoperabilidade superior e consequente redução de custos na implementação do sistema.

2.3 Automatização de Testes de Aceitação

Os testes de aceitação têm como principal objetivo assegurar que o sistema corresponde às expectativas dos utilizadores, garantindo que se encontra preparado para ser utilizado de forma eficaz, respeitando os requisitos previamente especificados[29].

Dado o caráter crítico do projeto *eMudar*, o desenvolvimento é sempre acompanhado por um caderno de testes de aceitação elaborado com base nos requisitos do sistema. Estes, são executados manualmente antes da entrega de um módulo e sempre que lhe sejam feitas alterações relevantes.

Ao longo dos últimos anos tem-se vindo a assistir a um aumento de consciencialização da importância e valor da qualidade de software[30]; no entanto, dadas as frequentes limitações financeiras, torna-se fundamental que esta etapa do *SDLC* seja não só eficaz, mas também o mais eficiente possível. Assim, um dos principais desafios que se coloca nesta área é a redução da elevada taxa de testes manuais ainda utilizados[30]. Em sistemas complexos como o *eMudar*, em constante crescimento e mutação, a presença de testes puramente manuais torna-se numa deseconomia de escala, o que pode levar à sobrecarga dos elementos responsáveis pela garantia de qualidade. Dadas as limitações humanas e financeiras que esta etapa manifesta, é frequente existir um aumento do *TTM* ou conduzir à entrega de um produto insuficientemente testado.

Para colmatar estas dificuldades é essencial que exista um conjunto de testes automáticos que auxiliem no processo de garantia de qualidade, libertando os responsáveis desta área para tarefas de conceção e realização de outro tipo de testes, contribuindo para uma maior qualidade e confiabilidade no sistema[30].

Antes de investir no processo de automatização de testes é importante analisar as necessidades e as particularidades do projeto de forma a garantir a escolha da ferramenta mais adequada. O grupo do *LAWST* (Los Altos Workshops on Test Automation) refere um conjunto de 27 questões para auxiliar na tarefa de formalização dos requisitos para um projeto de automatização de testes[31]. No entanto, atendendo a que a tarefa de automatização dos testes é um objetivo opcional do estágio, e não uma componente nuclear, o esforço dedicado nesta

tarefa focou-se na realização de uma prova de conceito em detrimento da conceção de um projeto de automação propriamente dito.

São muitas as soluções existentes no mercado para automatização de testes *black-box* [32][29][33], pelo que o universo de comparação incidiu sobre ferramentas que suportem o teste a sistemas *web-based*, através da simulação da ação do utilizador a nível do *browser*. Fica igualmente fora do âmbito da análise o estudo da execução de testes recorrendo a *browsers headless* já que, apesar de proporcionarem uma execução potencialmente mais rápida, considera-se que a sua utilização não é representativa do real[34].

Com base nestas necessidades e tendo por referência a lista de questões elencadas anteriormente, definiram-se os seguintes critérios para a escolha de uma ferramenta:

Suporte de *browsers*
Tem de suportar os *browsers* suportados atualmente no projeto, nomeadamente o *Internet Explorer 11* e o *Mozilla Firefox*. Dado que são ambos amplamente utilizados no mercado e tendo-se verificado que todas as ferramentas consideradas os suportavam, optamos por não incluir esta categoria na tabela 2.6.

Sistemas Operativos Compatíveis
Tal como no critério anterior, a ferramenta escolhida deve suportar os sistemas operativos em utilização no projeto, concretamente: *Windows* e distribuições de *Linux* focadas no mercado empresarial como o *Red Hat Enterprise*.

Custos de Aquisição
Face às contingências orçamentais do projeto, não se encontram previstas despesas para a aquisição de licenças deste tipo, pelo que é dada preferência a soluções gratuitas. São também consideradas soluções comerciais, com o objetivo de avaliar as suas vantagens face às alternativas gratuitas, procurando-se destacar a produtividade na sua utilização. Nesta categoria identificam-se três tipos de licenças:

- **Individuais** - São associadas a uma pessoa individual (tipicamente ao seu computador) e incluem, por norma, um suporte comercial durante o seu período de vigência.
- **Flutuantes** - Também conhecida como licença concorrente, não se encontra porém associada a nenhuma pessoa individual, definindo-se em alternativa o número de acessos simultâneos à ferramenta dentro de uma organização.
- **Execução** - Esta variante, habitualmente menos dispendiosa, destina-se apenas à componente que permite realizar a execução dos testes, revelando-se de grande utilidade através de mecanismos de integração contínua. De salientar que nem todas as soluções em análise exigem a utilização de licenças deste tipo, podendo as licenças flutuantes ser aproveitadas para este efeito (sempre que exista alguma *slot* disponível).

A nível de duração poderão realçar-se dois tipos de licenças:

- **Temporárias** - As licenças deste tipo são limitadas no tempo, necessitando de ser renovadas de forma a permitir a utilização do *software*. Normalmente a sua renovação permite o acesso a um período equivalente de suporte comercial, atualização e manutenção.
- **Perpétuas** - Ao contrário da modalidade anterior, estas licenças não possuem uma data de expiração. No entanto, o contrato de suporte comercial / atualização / manutenção apresenta habitualmente uma duração de 12 meses, ficando a sua renovação ao critério do cliente.

Linguagem de *scripting* e Extensibilidade

Ainda que a maior parte das ferramentas disponibilizem métodos para a gravação de testes através das ações de um utilizador, por vezes é necessário estende-las através de uma lógica adicional procurando satisfazer as necessidades do sistema em teste (*SUT*). Assim, destaca-se a extensibilidade enquanto fator de elevada importância, sendo desejável que a sintaxe suportada pela *framework* não obrigue a equipa a aprender uma nova linguagem de programação, devendo favorecer-se

o suporte à linguagem Java ou a esquemas alternativos de alto nível e de fácil aprendizagem.

Gravação de Testes

Este tipo de funcionalidade é particularmente interessante pois favorece o aumento da produtividade ao permitir automatizar a ação de um utilizador real. Porém é importante ter também em consideração que esta abordagem pode resultar num aumento de custos de manutenção[7][35], uma vez que contribui para a criação de testes frágeis (que falham devido a alterações no sistema)[36].

Adequação para *non-coders*

Neste critério pretende-se avaliar a possibilidade das *frameworks* poderem ser utilizadas por elementos da equipa sem conhecimentos de programação. Para isso, é desejável a existência de uma interface gráfica que facilite o acesso às principais operações e funcionalidades como a gravação de testes.

Documentação Técnica Oficial

De forma a garantir uma curva de aprendizagem baixa, é desejável que a ferramenta escolhida possua um extenso conjunto de documentação e guias orientadores, que facilitem a iniciação à mesma, favorecendo também a auto-aprendizagem. Nesta categoria considera-se apenas a documentação que se encontra acessível através do endereço oficial da ferramenta, independentemente do seu formato: texto, imagens, vídeo-guias, etc.

Na avaliação desta categoria consideraram-se quatro níveis distintos:

- Documentação rica (composição de texto, imagens e vídeo-guias), com um elevado grau de completude e facilmente acessível e adequada para iniciantes;

[7] Automated Testing != Record-Playback Tool - `http://goo.gl/r2rwhm`

- ◕ Documentação rica (composição de pelo menos dois dos seguintes tipos: texto, imagens e vídeo-guias) e com um satisfatório grau de completude e acessibilidade;
- ◑ Documentação satisfatória de acessibilidade relativa, apresentando algumas limitações;
- ◔ Conteúdo reduzido e/ou desatualizado de difícil acessibilidade.

Mecanismo de Relatórios

É importante que na presença de uma falha num teste se consiga identificar rapidamente o defeito que a originou. Para isso, é fulcral que existam relatórios detalhados que permitam um rápido diagnóstico e resolução do problema encontrado. Saliente-se que algumas ferramentas incluem métodos de captura de imagens de ecrã, tanto periódica (de forma automática), como manualmente. Outras, permitem ainda exportar o relatório para formatos úteis como: xUnit[37], Word, PDF, etc.

Na avaliação desta categoria foram considerados quatro níveis distintos:

- ● Suporte nativo à criação de relatórios detalhados, incluindo imagens e possibilitando a sua exportação para outros formatos;
- ◕ Suporte nativo à criação de relatórios com um bom nível de detalhe, que pode incluir imagens mas sem mecanismos internos de exportação;
- ◑ Suporte nativo à criação de relatórios com um bom nível de detalhe mas sem a inclusão de capturas do ecrã da aplicação.
- ◔ Ausência de suporte nativo à criação de relatórios, podendo ser colmatado recorrendo a ferramentas ou *plugins* de terceiros.

Suporte

Ainda que não seja necessária a existência de suporte comercial, é desejável que

a ferramenta escolhida seja suportada por uma ampla comunidade, apresentando um desenvolvimento ativo e um grau de estabilidade aceitável.

Suporte ao paradigma *Data-Driven*

Este modelo assenta na ideia de separar a lógica de um teste, dos dados que lhe são necessários. Este conceito permite que um teste possa ser facilmente reutilizado, recorrendo apenas a dados de entrada diferentes, evitando-se assim a duplicação de código[36][38].

Suporte ao *Gherkin*

O *Gherkin* é uma *Business Readable, Domain Specific Language*[8] criada com o objetivo de permitir documentar e especificar um *software* focando-se no seu comportamento e não na sua implementação[9]. Neste critério, mais importante que o suporte à linguagem propriamente dita, pretende-se avaliar o suporte à filosofia subjacente e oriunda dos princípios do *Behavior-driven development*. Considera-se que este tipo de práticas contribui para uma legibilidade melhorada, favorecendo o seu entendimento por pessoas não pertencentes à área tecnológica, tais como: como analistas e membros da equipa de qualidade de *software*.

Integração Contínua

Para que uma suite de testes automáticos seja adotada de forma natural e eficaz no *SDLC* da empresa é conveniente que seja facilmente interoperável com ferramentas de integração contínua como o *Jenkins* (ferramenta para o efeito em utilização no projeto), permitindo assim que a execução dos testes automáticos possa vir a ser desencadeada de forma automática. A nível da classificação foi atribuído a designação "sim"àquelas que permitem integração nativa com o *Jenkins*.

Nota 1: As informações presentes na tabela 2.6 foram obtidas através de informação publicamente acessível na *web*, de origem em texto, imagem e vídeo, podendo estar sujeita a alterações futuras.

[8] BusinessReadableDSL - `http://martinfowler.com/bliki/BusinessReadableDSL.html`

[9] Documentação do *Gherkin* - `http://docs.behat.org/en/v2.5/guides/1.gherkin.html`

Nota 2: Salvaguarda-se que a análise efetuada foi focada nas necessidades atuais do projeto, podendo deste modo não evidenciar certas virtudes de algumas das ferramentas visadas, dada a sua irrelevância no contexto do projeto. Como exemplo destaca-se o suporte ao teste de tecnologias alternativas (*mobile*, aplicações *Desktop*, etc), e a mecanismos de gestão de defeitos e mapeamento de requisitos, etc.

Nota 3: Salienta-se que as categorias *Documentação Técnica Oficial* e *Mecanismo de Relatórios*, não assentando em critérios totalmente objetivos, podem levar a uma avaliação parcial.

Nota 4: De referir que a solução *Tricentis Tosca Testsuite* foi também inicialmente considerada, uma vez que se encontra bem cotada e reconhecida no estudo da *Gartner*[32]. Porém, em virtude da dificuldade em encontrar informações técnicas detalhadas de forma a satisfazer todas as categorias consideradas, a mesma acabou por ser removida da nossa análise.

Tabela 2.6: Comparativo entre *frameworks* para a automação de testes *black-box*

	DevExpress TestCafe	HP Unified Functional Testing (QTP)	IBM Rational Functional Tester	Oracle Functional Testing	Ranorex	Robot Framework	Selenium	Borland Silk Test	SmartBear Test Complete	Telerik Test Studio
Sistemas Operativos	Win, Linux, Mac	Win	Win, Linux	Win, Linux[6]	Win	Win, Linux, Mac	Win, Linux, Mac	Win	Win	Win
Licença Individual	499$ / ano	2500$	7230$ [8]	1389€ /ano ou 6946€ [5]	1990€ [8]	gratuito	gratuito	-	2134€ [9]	2499$ [10]
Licença Flutuante	-	3500$	13900$ [8]	-	3490€ [8]	gratuito	gratuito	5 679 €	4810€ [9]	-
Linguagem de *scripting*	JScript	VB Script	Java, VB.NET	Java	C#, VB.NET	Python, Java, .NET	C#, Haskell, Java, JScript, Objective-C, Perl, PHP, Python, R, Ruby	Java, VB.NET	Python, VBScript, JScript, DelphiScript, C++Script, C#Scrip	C#, VB.NET
Gravação de Testes	sim	sim	sim	sim	sim	por terceiros[12]	sim	sim	sim	sim
Adequado para *non-coders*	sim	sim	sim	sim	sim	+/-[3]	não	sim	sim	sim
Documentação Técnica Oficial	◕	◔	◑	◕	●	●	◕	◔	●	●
Relatório	●	◕	◑	◕	◕	●	◔[11]	◕	●	●
Suporte	comercial (12m)	comercial (12m)	comercial (12m)	comercial 1528€ / ano	comercial (12m)	comunitário	comunitário e comercial (por terceiros)	comercial (12m)	n/a	comercial (12m)
Data-Driven	sim[1]	sim	sim	sim	sim	sim	sim (manualmente)	sim	sim	sim
Suporte a Gherkin	não	não	não	não	sim (plugin)	sim	sim	sim (plugin)	sim (plugin)	não
Integração Contínua	adaptável[14]	1800$ [2 8]	adaptável[13]	adaptável[7]	690€ [2 8]	sim	adaptável[13]	sim[4]	539€	349€

1 - Exemplo de utilização: `https://goo.gl/q15lKo`
2 - Custo de uma licença apenas de execução (*runtime*)
3 - Usabilidade mais limitada dado que não possui uma interface gráfica nativa como outras ferramentas
4 - A licença flutuante permite apenas 1 execução concorrente. Cada *runtime* adicional custa 3246€
5 - Preços obtidos em: `https://goo.gl/oRDWvS`
6 - Nem todas as componentes estão disponíveis em Linux
7 - Adaptável ao *Jenkins* através de suporte genérico a *scripts* de linha de comandos: `http://goo.gl/ovM9od`
8 - Preços sem impostos
9 - Corresponde às soluções *TestComplete Platform* e *TestComplete Web*
10 - Corresponde à solução *TestStudio Web & Desktop*
11 - Suporte a relatórios através de 3rd party tools: `http://goo.gl/qXvQEO`
12 - Exemplo de ferramenta de suporte: `https://github.com/joao-carloto/FireRobot`
13 - Assumindo a presença de uma *framework* da família xUnit[37]
14 - Adaptável ao *Jenkins* sem custos adicionais: `http://testcafe.devexpress.com/FAQ`

Analisando a tabela 2.6 conclui-se que a maior parte das soluções no mercado estão sobretudo vocacionadas para sistemas Windows. No entanto, apesar deste sistema operativo ser utilizado no desenvolvimento do projeto *eMudar*, a nível de servidores e de integração contínua é desejável que a ferramenta a designar suporte distribuições de *Linux*.

Dado este requisito, sobram as soluções: *TestCafe*, *IBM RFT*, *Oracle FT*, *Robot Framework* e *Selenium*. Esta última deve ser excluída uma vez que não possui suporte nativo para a geração de relatórios e não se encontra vocacionada para *stakeholders* sem conhecimentos de programação. O *TestCafe* da *DevExpress*, ainda que seja considerado como uma das soluções comerciais mais económicas, está associado a uma licença individual periódica, o que aliado à sua extensibilidade limitada (apenas *JavaScript*) e à ausência de suporte ao *Gherkin*, se traduz numa fragilidade que deve resultar na sua exclusão.

Tanto as soluções da IBM como da Oracle possuem um elevado leque de funcionalidades que não se encontra detalhado nesta comparação; porém, apesar de satisfazerem grande parte dos requisitos identificados, os seus elevados custos de aquisição tornam difícil a sua aquisição. Deste modo, a solução passará pela utilização da *Robot Framework* para a criação de uma prova de conceito de automatização de testes no projeto *eMudar*.

Capítulo 3

Gestão de Projeto

Neste capítulo são referidos os principais aspetos de gestão utilizados durante o presente estágio, abordando-se a organização e alguns processos da equipa de projeto onde o estagiário foi integrado, processos de engenharia e o planeamento a ser seguido durante o estágio.

3.1 Projeto *eMudar*

3.1.1 Organização da Equipa

O projeto conta com pouco mais de trinta pessoas, o que favorece a sua divisão em sub-equipas, de forma a facilitar a sua gestão. A saber:

Gestão - Composta pelos responsáveis pela coordenação, monitorização e logística do projeto, contemplando também tarefas de caráter financeiro e burocrático;

Análise Funcional - Esta equipa é responsável pela elicitação e manutenção de requisitos junto do cliente;

Desenvolvimento - Existem duas equipas responsáveis pela análise técnica e implementação de novas funcionalidades solicitadas pelo cliente, recorrendo

à análise funcional já efetuada;

Manutenção Evolutiva - Responsável por manter os módulos que já foram aceites pelo cliente, e que já se encontram em produção. As suas funções passam não só pelo diagnóstico e correção de defeitos, mas também pela expansão do sistema através de pedidos de alteração solicitados pelo cliente.

Suporte - Garantem a assistência de segundo nível ao *BFA*, no decorrer das atividades diárias.

Salienta-se ainda que cada uma destas sub-equipas possui um líder responsável pela sua coordenação, resultando num modelo de controlo e monitorização mais distribuído, o que contribui para a redução do *overhead* da sua gestão.

3.1.2 Ambientes

Atendendo ao carácter crítico do sistema *eMudar* e ao elevado tamanho da equipa, existem diversos ambientes que são utilizados para auxiliar nas tarefas de desenvolvimento e de garantia de qualidade.

- **Local** - Consiste numa instância do *IBM WebSphere*, um *Java EE Application Server* que executa os módulos mais relevantes para o desenvolvimento;
- ***Development (Trunk)*** - Este ambiente é utilizado sobretudo pelas equipas de desenvolvimento, sendo alvo de alguns (<10) deployments diários. O seu potente *hardware* permite integrações mais rápidas do que em ambiente local;
- ***Test*** - Servidores utilizados para efeitos de garantia de qualidade pela equipa do *eMudar*, a diferentes *branches* de código;
- **Qualidade** - Este ambiente constitui o primeiro nível de entrega de uma dada versão ao *BFA*, sendo usado pela sua equipa de Garantia de Qualidade para efeitos de validação;
- **Pré-Produção** - Este ambiente é o mais parecido, a nível de *hardware* e de configurações, com o ambiente de produção, sendo usado pela equipa de Garantia de Qualidade do *BFA* para validar se uma dada versão pode ser

instalada em produção. Habitualmente as versões instaladas neste ambiente, foram já alvo de aceitação no ambiente de qualidade;

- **Produção** - Ambiente final utilizado pelo cliente nas operação diárias;

Salienta-se que a nível do Sistema Legado *BANKA*, apenas existem duas variantes, a de produção e uma alternativa que é utilizada para todos os outros ambientes referidos, dificultando assim que se obtenha um isolamento completo entre ambientes.

3.1.3 *Clarification Request*

Atendendo ao processo de desenvolvimento utilizado no projeto, é esperado que as tarefas de Análise Funcional, Implementação e Garantia de Qualidade e Testes sejam efetuadas por pessoas distintas. Esta prática contribui para uma maior variabilidade das pessoas que estão envolvidas em cada componente do sistema, favorecendo a deteção precoce de eventuais situações inusitadas.

Não obstante, é necessário que as pessoas envolvidas apresentem um espírito crítico de acordo com a sua experiência e responsabilidades profissionais, o que poderá, ocasionalmente, resultar em dúvidas. Neste tipo de situações, prevê-se a criação de um pedido de esclarecimento através da plataforma *JIRA*, atribuindo-o à pessoa ao responsável pela tarefa em causa. Para além de uma resposta, estes pedidos podem ainda levar a alterações de requisitos, de documentação, de código ou mesmo testes.

Saliente-se que esta prática visa assegurar que, não só as questões e respetivas repostas, ficam registadas e acessíveis a todos os membros da equipa, como também a eventuais ações que se venham a desenvolver, permitindo assim a sua correta rastreabilidade.

No contexto do presente estágio, este processo foi exercitado, em cinco ocasiões, em relação à análise funcional do Módulo de Relatórios, pois ainda que os requisitos tenham sido aceites pelo cliente e exista o cuidado de ser o mais explícito possível na sua definição, nem sempre é fácil garantir a sua completude.

3.1.4 *Handover*

Este processo consiste na transferência de conhecimento e delegação de responsabilidades sobre um módulo ou CR de médias/elevadas proporções, entre uma das equipas de desenvolvimento e a equipa de manutenção. Ocorre no seguimento de uma passagem a produção, após a aceitação por parte do cliente. Este processo visa assegurar que a equipa de suporte e manutenção obtém o conhecimento necessário para manter o módulo, dotando-a de autonomia suficiente para que no futuro possa fazer melhorias e corrigir eventuais defeitos, libertando a equipa de desenvolvimento para a implementação de novas tarefas.

3.2 Processos de Engenharia

Nesta secção pretende-se dar a conhecer alguns dos processos mais importantes no âmbito do projeto e do estágio.

3.2.1 Ciclo de Vida

De forma a conceber um sistema crítico que satisfaça as necessidades do cliente e permita uma evolução compatível com a do negócio, o projeto *eMudar* segue um Ciclo de Vida Incremental. Estes incrementos, podem surgir tanto devido a novas funcionalidades, como devido a *Change Requests (CRs)* solicitados pelo cliente.

Na figura 3.1 é possível observar uma ilustração do ciclo de vida referido, baseado em pequenos incrementos que envolvem as diversas etapas habituais de um ciclo de vida em cascata (*waterfall*)[39]. No entanto, atendendo ao caráter crítico do sistema em questão, cada um destes incrementos exercita uma variação do modelo em cascata, denominada modelo em V, que é ilustrado na figura 3.2.

Do lado esquerdo da figura 3.2, observam-se algumas das múltiplas etapas que constituem o tradicional modelo de desenvolvimento em cascata[39]. Na coluna central da imagem, encontram-se os diferentes tipos de teste que podem ser elaborados para verificar cada uma das etapas que intervêm no desenvolvimento. Por

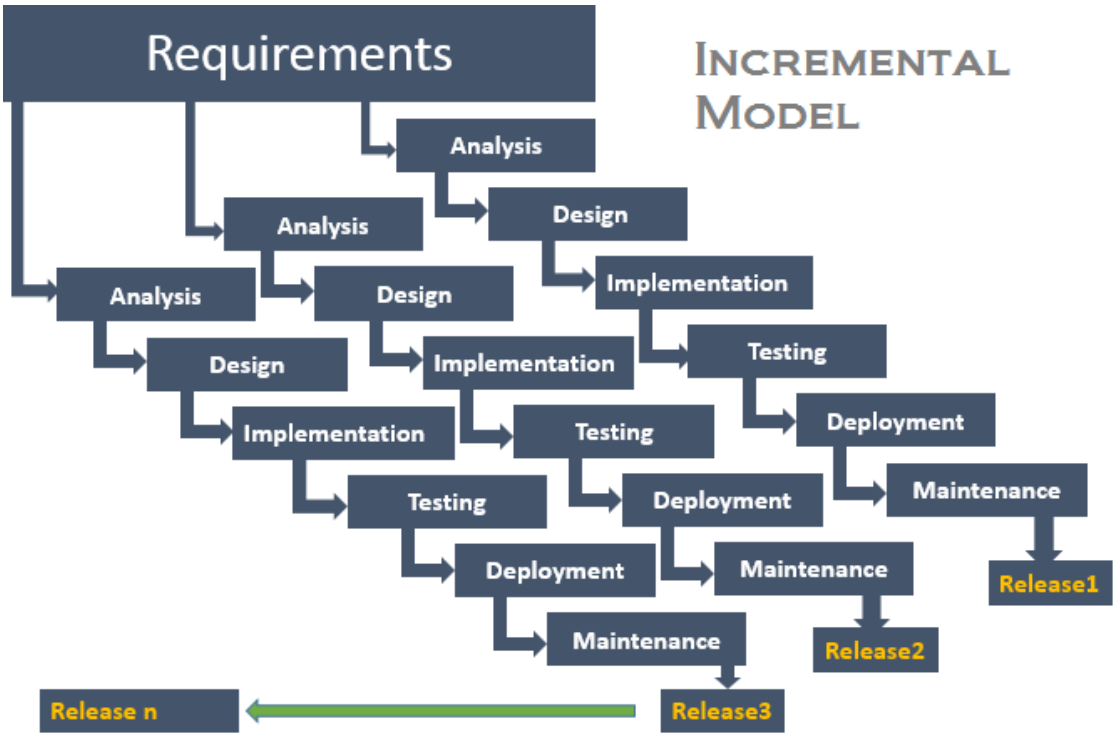


Figura 3.1: Esquema ilustrativo de um Ciclo de Vida Incremental[2]

fim, do lado direito, encontram-se representadas as diversas fases de verificação que serão executadas recorrendo aos testes previamente especificados. Este tipo de modelo, promove uma maior rastreabilidade entre cada uma das fases do desenvolvimento e os respetivos testes, garantindo que cada fase é verificada de forma independente e com base na sua especificação[40].

Dada a sua complexidade, este tipo de modelo torna-se dispendioso, pelo que a sua aplicação poderá ser ajustada de acordo com a importância e impacto das tarefas em implementação.

[2]Referência para a imagem: http://testingfreak.com/wp-content/uploads/2015/02/incremental.png

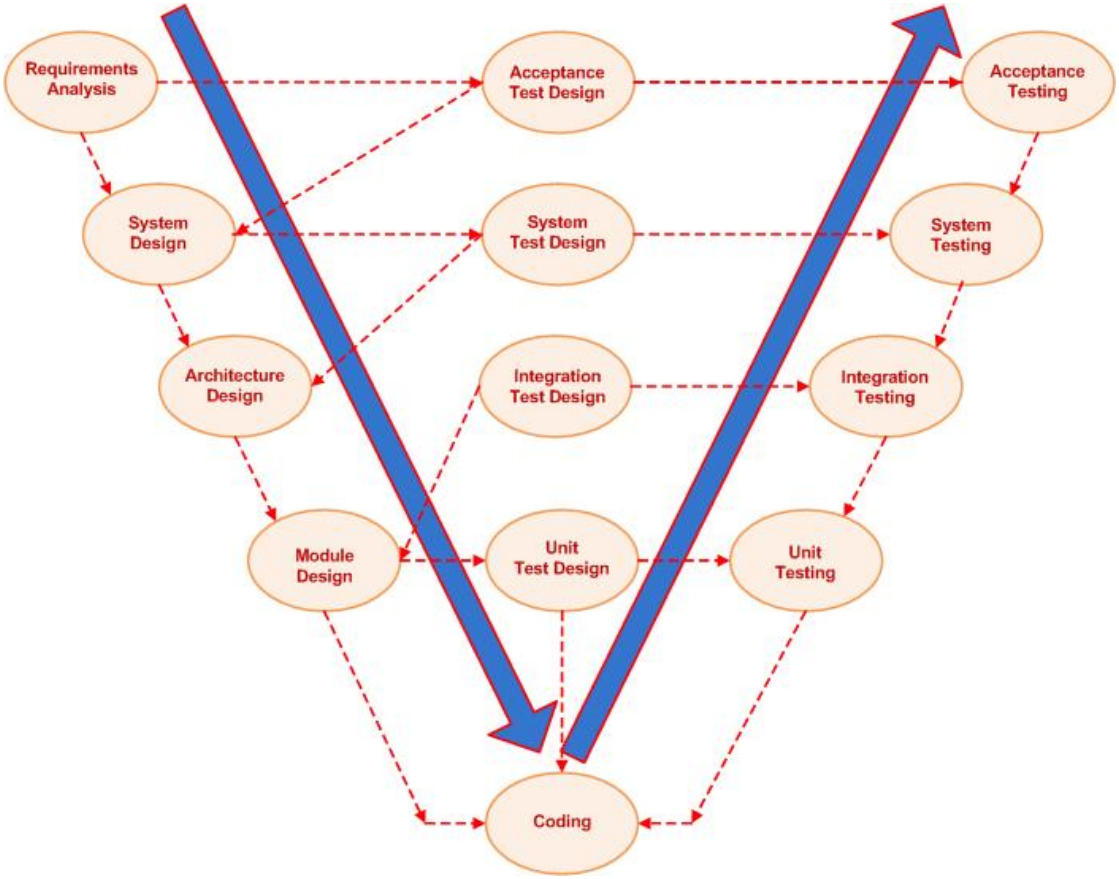


Figura 3.2: Esquema ilustrativo do modelo de Desenvolvimento em V[3]

3.2.2 Gestão de Riscos

Entende-se por risco, todo e qualquer fator que possa resultar em consequências negativas no futuro[41]. Em Engenharia de Software, é essencial que se proceda a uma gestão de riscos, isto é, à aplicação sistemática de procedimentos e práticas, com vista à identificação, análise, priorização e controlo de riscos[41], de forma a minimizar o impacto da sua eventual ocorrência[42].

Neste ponto é descrita a notação utilizada na avaliação dos riscos que foram identificados para as principais componentes que constituem este estágio, que são frequentemente expressos em função do seu Impacto, Probabilidade de Ocorrência e

[3]Referência para a imagem: https://upload.wikimedia.org/wikipedia/commons/9/96/V-model.JPG

Janela Temporal[42][41].

Relativamente ao impacto, consideraram-se os seguintes níveis:

- **Catastrófico** - Compromete seriamente o sucesso do estágio e/ou a qualidade do produto entregue;
- **Crítico** - Poderá comprometer o sucesso do estágio e/ou a qualidade do produto entregue, caso não seja despendido tempo adicional para a sua recuperação;
- **Marginal** - Não constitui uma ameaça ao sucesso do estágio, implicando, no entanto, ajustes a nível do planeamento.

No que diz respeito a probabilidades de ocorrência, consideraram-se as três seguintes categorias:

- **Alta** - superior a 70%;
- **Média** - entre os 40% e os 70%;
- **Baixa** - inferior a 40%.

Priorização
Recorreu-se à técnica *Pareto Top N*[42] de forma a proceder à seleção dos riscos a mitigar, recorrendo à conjugação dos parâmetros de impacto e probabilidade de ocorrência. Considerou-se que, dado o contexto do projeto *eMudar*, seria importante mitigar todos os riscos com probabilidade de ocorrência igual ou superior a 40% (média/alta) e cujo impacto se encontrasse entre os níveis crítico e catastrófico.

3.2.3 Análise Funcional

Atendendo à distância que existe ao cliente, a elicitação de requisitos é habitualmente elaborada recorrendo a pedidos feitos por escrito, complementados por tele-conferências e pedidos de clarificação (ver secção 3.1.3). A concretização deste

processo começa pela elaboração de protótipos de média fidelidade, sendo posteriormente utilizados para a conceção de um *storyboard*, uma sequência de *mockups* que permitem simular o funcionamento do sistema. Ao serem validados pelo cliente, estes, são utilizados para formalizar uma Especificação de Requisitos, segundo um estilo textual declarativo. A sua posterior aprovação permite desbloquear a conceção de Testes de Aceitação, bem como a sua Análise Técnica.

3.2.4 Estimação

De forma a efetuar um planeamento útil e rigoroso, é necessário utilizar um processo de estimação confiável e bem definido, contribuindo para a formulação de objetivos realistas e tangíveis. Desta forma, foi seguida uma abordagem de estimação *bottom-up*, incidindo nas tarefas do nível de granularidade mais fino da *WBS*, recorrendo à técnica *Three-Point Estimation*. Esta consiste na especificação de três estimativas para cada tarefa:

- ***Best Case*** - duração prevista que demorará a completar a tarefa se tudo correr bem (otimista);
- ***Most Likely*** - duração prevista que demorará a completar a tarefa no cenário mais provável;
- ***Worst Case*** - duração prevista que demorará a completar a tarefa no pior caso possível (pessimista).

A estimativa final é depois calculada através da fórmula:

$$\text{estimativa} = (\textit{Best Case} + 4 \times \textit{Most Likely} + \textit{Worst Case}) \,/\, 6$$

Em casos de maior incerteza a fórmula pode ser ajustada, de forma a atribuir um peso superior à componente de *Worst Case*.

Destaca-se ainda que estas estimativas devem ser acompanhadas dos pressupostos considerados na sua elaboração, facilitando assim a sua avaliação por outras pessoas.

Para complementar esta abordagem, utilizou-se também a técnica *expert judgement*, que recorre a um membro com elevada experiência no projeto, para validar

as estimativas efetuadas, garantindo correção do racional e dos pressupostos indicados.

3.2.5 Análise Técnica

Este processo precede a fase de codificação e consiste na análise, sob o ponto de vista técnico, das tarefas a serem implementadas, seja no âmbito de novos desenvolvimentos, ou de um *CR*, contemplando as três fases seguintes:

Decomposição segundo uma estrutura analítica do projeto

Nesta primeira etapa, efetua-se uma análise crítica das tarefas a implementar, com vista à sua problematização e expansão, proporcionando um melhor entendimento do problema. Este conhecimento é de seguida utilizado para decompor o trabalho segundo uma Estrutura Analítica de Projeto (*WBS*), promovendo uma representação estruturada e hierárquica do mesmo, facilitando os processos de estimação, planeamento e monitorização;

Análise de Impacto

Enquanto que na etapa anterior se obtém uma visão detalhada das subtarefas que decompõem o problema, nesta, avalia-se o impacto que estas alterações poderão vir a ter no projeto. Semelhante a uma análise de riscos, procura garantir que as alterações efetuadas não irão comprometer outras funcionalidades, ou, se necessário, que essas implicações são consideradas durante a fase de estimação;

Estimação

Esta etapa é feita de acordo com o processo descrito na secção 3.2.4.

3.2.6 Reuniões

A nível do projeto *eMudar* são feitas reuniões gerais, com toda a equipa, de três em três semanas, sendo feita uma revisão geral do trabalho desenvolvido desde a última reunião, abordando as metas e objetivos desejáveis até à próxima reunião,

assim como aspetos relevantes como riscos e dados financeiros. Atendendo à dimensão da equipa, considera-se que este tipo de reuniões são fundamentais para a garantia de uma maior integridade conceptual por parte dos diversos elementos que a constituem.

Durante o estágio ocorreram também múltiplas reuniões informais com o **orientador da empresa** de forma a adquirir um contexto superior acerca do projeto e das práticas em vigor, a resolução de dúvidas e a validação da abordagem a desenvolver. Para além disso, foi também realizado um ponto de situação, com periodicidade mensal ou logo após a conclusão de uma etapa do estágio, onde era revisto o planeamento e o trabalho desenvolvido.

Por fim, em relação à componente curricular do estágio, foram agendadas reuniões de periodicidade mensal com o **orientador do DEI**, permitindo o acompanhamento do trabalho desenvolvido e a revisão do planeamento e do relatório de estágio.

3.2.7 Versionamento de Código

A base de código do projeto é mantida através de um sistema de controlo de versões, o SVN[4], sendo as novas funcionalidades implementadas recorrendo a *feature-branches*, derivações da árvore principal que permitem um desenvolvimento isolado.

3.2.8 Versionamento de Base de Dados

De forma a garantir um mapeamento mais rigoroso entre a versão da base de código do sistema *eMudar*, e das suas bases de dados, é necessário proceder, de forma análoga, ao versionamento das mesmas, garantindo um controlo e manutenção

[4]Página oficial do Apache™ Subversion: `http://subversion.apache.org`

superiores. Desta forma, é utilizada a ferramenta *Flyway*[5] para gerir a versão da camada de dados.

3.3 Planeamento do Estágio

O primeiro semestre que compõe o estágio pressupõe uma alocação de 40%, o que corresponde a dezasseis horas semanais, concretizando-se em dois dias por semana na empresa. O resumo das atividades desenvolvidas no primeiro semestre pode ser consultado no diagrama de *Gantt* representado na figura A.1 em apêndice.

Durante o segundo semestre que compõe o estágio, a alocação foi de 100%, o que corresponde a quarenta horas semanais, concretizando-se em cinco dias por semana na empresa. O planeamento efetuado para o segundo semestre pode ser consultado no diagrama de *Gantt* representado na figura A.3 em apêndice.

[5]Página oficial do *Flyway*: `https://flywaydb.org`

Capítulo 4

Desenvolvimento - Módulo de Relatórios

Neste capítulo são apresentadas as diversas etapas que constituem a fase de desenvolvimento do estágio. Desta forma, no âmbito do *Módulo de Relatórios* do *eMudar*, pretendemos fazer uma análise dos principais riscos que poderão afetar esta fase, abordando de seguida tópicos como a Análise Técnica, Conceção da Arquitetura e do Modelo de Dados, terminando com uma descrição da implementação do referido módulo bem como das suas principais particularidades.

4.1 Análise Funcional

A *Análise Funcional* deste módulo foi elaborada pela respetiva equipa do projeto, tendo sido aceite pelo cliente. O documento pode ser consultado no apêndice B.

Não obstante, foi feita uma revisão integral da especificação de requisitos no âmbito da ótica de garantia de qualidade, podendo ser consultada na secção 5.2.

4.2 Análise Técnica

De acordo com o processo descrito na secção 3.2.5, esta etapa precede a fase de implementação, e consiste na análise dos requisitos com vista à obtenção do contexto do módulo em causa, permitindo efetuar a identificação de condicionantes, a análise de impacto, a conceção da arquitetura e do modelo de dados da componente.

Figura 4.1: Caracterização de um relatório

Na figura 4.1 apresentam-se os diferentes tipos de caracterização de um relatório identificados ao abrigo desta tarefa. Encontram-se representados a verde, dois aspetos que podem ser utilizados na definição de um relatório, a saber:

Visualização - Um relatório diz-se genérico sempre que puder reutilizar um mecanismo base de visualização de dados de forma tabular. Pelo contrário, caso seja necessária uma visualização mais complexa, recorrendo a outro tipo de elementos gráficos e/ou interação com o utilizador, diz-se que o relatório é personalizado, uma vez que implica a extensão do mecanismo genérico de forma a satisfazer as suas particularidades.

Obtenção dos dados - Sempre que a geração de um relatório tenha por base uma operação leve e simples ao nível da base de dados, o mesmo pode ser

obtido de forma imediata. Por outro lado, quando os seus dados implicarem operações mais pesadas e/ou requeiram alguma etapa adicional de pré-processamento / enriquecimento, o mesmo não poderá ser obtido em tempo real, devendo ser, em alternativa, computado regularmente. Nesta situação, a periodicidade da geração deve ser considerada caso-a-caso, destacando-se a relação de compromisso entre as necessidades do negócio e o peso computacional das operações operações em causa.

Tabela 4.1: Análise das características dos diferentes relatórios previstos

Análise de Ecrãs	Visualização	Obtenção de dados	Pré-processamento	Persistência de dados	Nº de colunas
Processos atribuídos ao Administrador de Sistema	personalizada	imediata	○	-	7
Acessos x Digitalização x Impressão	personalizada	periódica	●	*schema* genérico	10
Processos em situação de Falha	genérica	imediata	○	-	2
Relatório de SMSs	genérica	periódica	○	vista materializada	2
Lista de respostas correctas negativas (Cartões)	genérica	periódica	○	vista materializada	5
Lista de respostas correctas negativas (Cheques)	genérica	periódica	○	vista materializada	5
Lista de respostas correctas negativas (BFA Net)	genérica	periódica	○	vista materializada	5
Lista de SMS's cujo envio falhou...	genérica	periódica	○	vista materializada	5
Processos cheques visados abertos/activos	genérica	periódica	●	*schema* genérico	5
Processos anulados e motivos de cancelamento	genérica	periódica	●	*schema* genérico	9

Na tabela 4.1 observa-se a avaliação feita para cada relatório com base na caracterização definida anteriormente na figura 4.1. Dado que são abordados diferentes tipos de aspetos, como camada de apresentação, modelo de dados e obtenção dos mesmos, a análise a cada um destes elementos é concretizada ao longo das secções seguintes.

4.3 Condicionantes

De acordo com a especificação de requisitos do Módulo de Relatórios, identificaram-se duas necessidades de extrema relevância para a sua implementação, a saber: a

extensibilidade e a modificabilidade.

Extensibilidade

Um módulo ou sistema considera-se extensível quando é possível incrementá-lo facilmente, sem afetar outras partes do mesmo[43]. Mais do que desejável, este atributo de qualidade é cada vez mais crucial para fazer face à constante mutação das necessidades do mercado.

No Módulo de Relatórios pretende-se que seja simples incluir novos relatórios sem que isso afete os já existentes. Desta forma, podem salientar-se dois cenários distintos de extensibilidade:

- **Relatórios Simples** - Um relatório desta categoria possui uma **visualização** baseada apenas em dados tabulares (visualização genérica) e pode ser computado de forma imediata dado o reduzido impacto a nível da sua **obtenção de dados**. Face a estas características, este tipo de relatórios deve poder ser criado recorrendo apenas a uma *query SQL*, dispensando assim alterações de código e o consequente *redeployment*;
- **Relatórios Personalizados** - Perante qualquer relatório que necessite de uma rotina de pré-processamento de dados e/ou de uma visualização mais complexa, com eventual possibilidade de interação, deve ser possível tirar-se partido da base de código existente. Pretende-se assim que o esforço dedicado à criação destes relatórios seja focado nas suas particularidades, sem desperdício de tempo com aspetos transversais, como: o acesso às bases de dados, o acesso à *BPMS*, a serialização de dados, configurações, entre outros.

Modificabilidade

A modificabilidade é a capacidade de um componente ou sistema ser alterado, de forma simples e eficiente, a nível de custo[43]. Atendendo a que a fase de manutenção do *SDLC* constitui cerca de 40% a 80% do custo de um *software*[44], este atributo de qualidade torna-se imprescindível para viabilizar o desenvolvimento de *software*, sendo também determinante para assegurar a longevidade de um sistema.

São muitas as táticas que podem ser adotadas com vista à promoção de uma maior modificabilidade de uma componente ou sistema[45], destacando-se:

Separação de responsabilidades - Deve tirar partido da arquitetura *Orientada a Serviços* utilizada no projeto, promovendo uma divisão clara das responsabilidades visadas no módulo e contribuindo para a redução dos custos da alteração de uma componente do mesmo.

Aumento da coesão - Deve-se favorecer a utilização de abstrações com vista à prevenção da duplicação de código, facilitando a modificação de componentes de caráter transversal.

Redução do acoplamento - Esta estratégia fomenta a restrição dos canais de comunicação entre componentes, determinando que uma camada lógica deva apenas comunicar com as que lhe são contíguas. Por outro lado, artifícios como o encapsulamento do acesso a serviços contribui positivamente para a concretização da mesma.

4.4 Análise de Riscos

No âmbito da fase de implementação do estágio e de acordo com o processo especificado na secção 3.2.2, podemos enumerar um conjunto de riscos que foram considerados mais relevantes nesta etapa:

1. A incorreta configuração ou eventual problema nas Rotinas que geram os relatórios são de difícil deteção. Os relatórios que dependem destas rotinas não são atualizados, exibindo assim dados desadequados.
2. A camada gráfica tem de ser implementada através da *framework* ZK[1], na qual o estagiário não tem qualquer experiência. As tarefas propostas podem sofrer atrasos, comprometendo o desempenho do estágio.
3. O ambiente de desenvolvimento é complexo e pesado, não permitindo um *deployment* célere. O esforço previsto para o desenvolvimento pode revelar-se superior ao esperado, comprometendo o desempenho do estágio.
4. A etapa de desenvolvimento só pode ser iniciada quando as *queries* de obtenção dos dados para os relatórios forem disponibilizadas pelo responsável da camada de dados do projeto. O atraso da sua disponibilização prejudica o planeamento proposto, afetando igualmente o desempenho do estágio.

Tabela 4.2: Matriz de Probabilidade-Impacto de riscos identificados na componente de implementação

Probabilidade	Impacto		
	Marginal	Crítico	Catastrófico
Alta		2,3	
Média		4	1
Baixa			

[1]Página oficial da *framework ZK*: `https://www.zkoss.org`

Mitigação

1. Deve assegurar-se a devida documentação da configurarão das rotinas, garantindo a correta renovação dos dados associados às mesmas. Assim, deve ser criado um conjunto de *scripts* com vista a facilitar o seu correto agendamento. Devem também ser introduzidas marcas temporais (*timestamps*) em todos os ecrãs de forma a facilitar o diagnóstico de problemas na obtenção dos dados. Na eventualidade de ser aberto um relatório, cujos dados tenham expirado, esse evento deverá ser registado internamente e a marca temporal associada ao relatório deve ser representada a vermelho, exibindo uma *tooltip* indicando que o relatório se encontra expirado - o que constitui um mecanismo de *feedback*

2. De forma a considerar o tempo necessário para aprendizagem da *framework*, a técnica de estimação utilizada (*3 point estimation*) deve ser ajustada de forma a atribuir um peso de 50% à componente de "Pior Cenário".

3. Deve ser utilizado o *plugin JRebel* para a *JVM*, de forma a aumentar a produtividade ao acelerar o processo de *deployment*, através da técnica de *hot swapping* que consiste no recarregamento, em memória, de recursos como: código Java, HTML, CSS, etc. Para lidar com a volatilidade das estimativas e dada a complexidade do projeto, sobretudo ao nível de integração de componentes, a técnica de estimação utilizada deve ser adaptada tal como no risco anterior.

4. Deve ser feito um acompanhamento constante para que as *queries*, que permitem obter os dados para os relatórios, sejam disponibilizadas ainda durante a fase de análise, assegurando a sua disponibilidade antes do início do desenvolvimento. No caso de ausência de dados, durante o desenvolvimento, esta etapa deve ser preemptada por outra tarefa sem precedências ativas, de forma a minimizar o impacto negativo no calendário proposto.

4.5 Arquitetura do Módulo

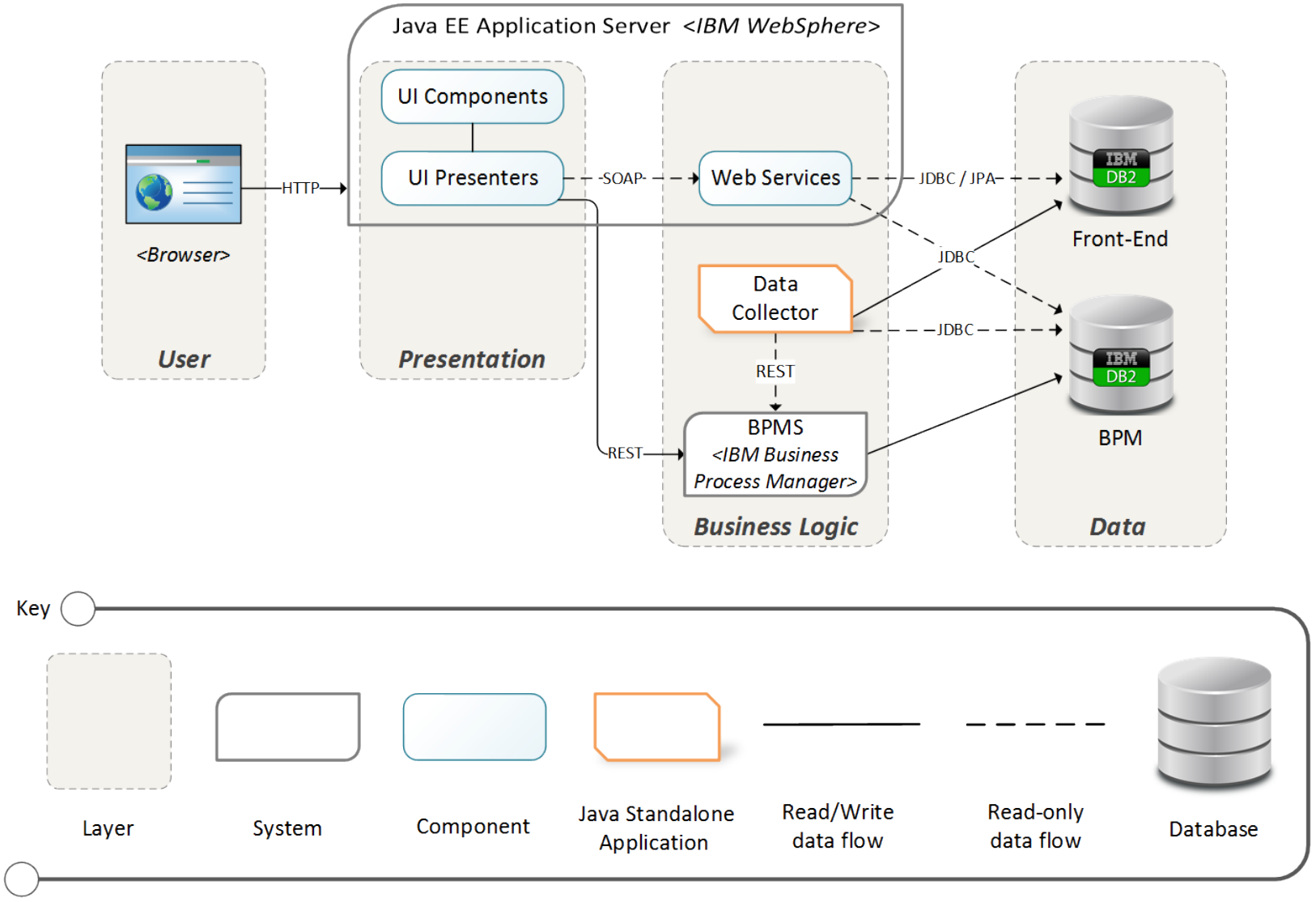


Figura 4.2: Vista de decomposição em camadas do Módulo de Relatórios

Na figura 4.2 é possível observar a forma como o Módulo de Relatórios está organizado e interage com os diferentes elementos de maior relevância para o estágio, sendo também ilustrado o fluxo de informação e as tecnologias utilizadas. Esta representação, ainda que não padronizada, assenta num estilo comum em camadas, em que as setas indicam a permissão de utilização entre elementos[46]. Concebida com o objetivo de fornecer uma visão geral do funcionamento do módulo é, sobretudo, útil enquanto referência para o auxílio no desenvolvimento e na manutenção futura do módulo. Destaca-se ainda que esta vista é, também, adequada para demonstrar certo tipo de atributos de qualidade como a modificabilidade e a manutenibilidade[45][46], dada a evidente separação de responsabilidades e o baixo acoplamento conseguido através da organização em camadas e das restrições de comunicação entre elas.

4.6 Implementação

A implementação deste módulo contempla cinco etapas distintas:

Conceção do Modelo de Dados - Nesta primeira tarefa pretende-se idealizar um modelo de dados que satisfaça as necessidades identificadas na especificação de requisitos (apêndice B) e durante a análise técnica (secção 4.2). A execução desta tarefa engloba a criação de um diagrama físico, representativo da estrutura do modelo de dados, bem como a geração de um *script DDL* que permita fazer a criação do respetivo modelo na base de dados;

Análise e obtenção dos dados - Seguidamente foram analisados os *scripts legados* que geram os antigos relatórios (conforme referido na secção 1.3.1), com vista à replicação da estratégia de obtenção dos dados. Saliente-se, conforme ilustrado na figura 4.1, que existem diversos graus de complexidade na obtenção dos dados de um relatório: uns, mais simples, obtêm-se recorrendo apenas a uma *query* à base de dados, outros, mais complexos, requerem uma etapa de enriquecimento, podendo envolver múltiplas *queries* e/ou interação com a *BPMS*;

Codificação de aplicação para o enriquecimento de dados - Devido a questões de desempenho, nem todos os relatórios podem ser gerados em tempo real, havendo uma relação de compromisso entre as necessidades do negócio e o impacto a nível de desempenho e usabilidade. Para colmatar esta situação, foi criada uma aplicação responsável pela obtenção e pré-processamento de dados, de forma periódica, de acordo com as necessidades identificadas na análise funcional do módulo;

Criação do Serviço de Relatórios - Respeitando os princípios da arquitetura do *eMudar* (*Arquitetura Orientada a Serviços* - SOA), deve ser criado um serviço responsável pela obtenção dos dados de um relatório e pela lógica subjacente ao mesmo, entre os quais se pode destacar: paginação, pesquisa, ordenamento de dados, entre outros;

Camada de Apresentação - Finalmente, foi elaborada a interface do utilizador que permitite a visualização de relatórios dentro da plataforma *eMudar*. Utilizou-se a *framework ZK* que se encontra em conformidade com *design-patterns* como o *MVC*, possibilitando a mitigação das condicionantes identificadas na secção 4.3.

4.6.1 Modelo de Dados

Para satisfazer as necessidades do Módulo de Relatório a nível da camada de persistência de dados, são abordados nesta secção dois pontos distintos, a saber: Vistas Materializadas e Definição de um *schema* Genérico.

Vistas Materializadas

Atendendo à caracterização de relatórios apresentada na figura 4.1 e às diferentes combinações de relatórios ilustradas na tabela 4.1, destacam-se, nesta secção, os relatórios cujos dados provêm de uma *query SQL* computacionalmente pesada, mas que não requerem nenhuma etapa adicional de enriquecimento de dados.

De acordo com a especificação de requisitos, 5 dos 10 relatórios previstos enquadram-se nesta categoria, pelo que se optou pela utilização de um *design pattern* utilizado frequentemente em *data warehousing*, denominado *Materialized View Pattern*[47].

Assim, tal como uma *vista* tradicional, uma *vista materializada* também se encontra associada a uma *query* que permite obter os seus dados; no entanto, a principal diferença prende-se com o facto dos dados obtidos serem persistidos numa entidade virtual, constituindo uma espécie de *cache*, que proporciona uma diminuição significativa do peso computacional de sucessivos acessos. A nível de extensibilidade, caso os relatórios assentem numa visualização genérica e sejam baseados nesta estratégia de armazenamento podem igualmente ser adicionados sem qualquer alteração a nível de código ou necessidade de *deployment* no servidor de aplicações.

Definição de um *Schema* Genérico

De forma a fazer face aos relatórios cuja obtenção implica uma etapa pesada de pré-processamento / enriquecimento de dados (consultar a figura 4.1), foi necessário desenvolver um modelo de dados que permitisse fazer o armazenamento dos dados deste tipo de relatórios, minimizando assim substancialmente a carga no seu acesso.

Durante a elaboração deste modelo concedeu-se especial atenção à condicionante da extensibilidade, identificada na secção 4.3. Apesar de todos os relatórios especificados nos requisitos do Módulo de Relatórios (apêndice B) assentarem numa visualização de dados de forma tabular, destaca-se que cada um tem o seu número e tipo de colunas, dificultando a especificação de um *schema* relacional único que suporte as necessidades de todos os relatórios.

Para colmatar esta limitação das *RDBMS* em lidar com dados heterogéneos[48], destaca-se que se torna mais apropriada a utilização de uma base de dados alternativa, baseada na filosofia *NoSQL*, concretamente na vertente *document-oriented*, destacando-se soluções como o MongoDB[1] e o Couchbase[2].

[1]Página oficial do MongoDB: `https://www.mongodb.com`
[2]Página oficial do Couchbase: `http://www.couchbase.com`

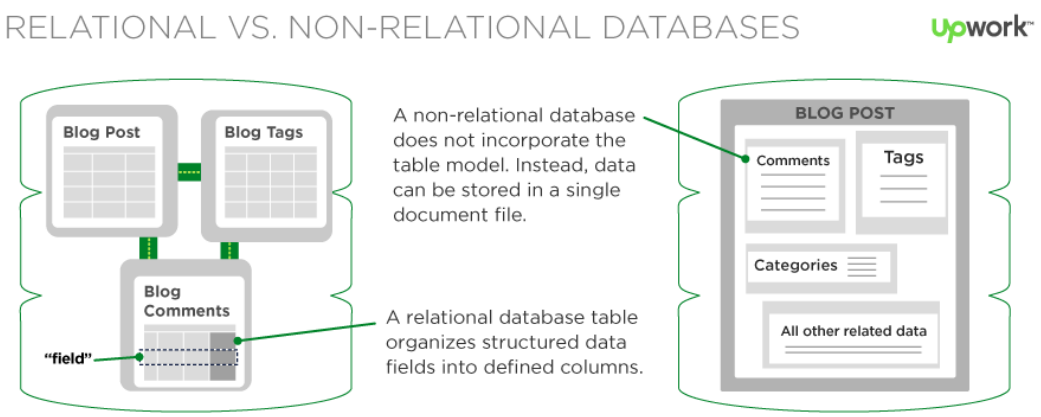


Figura 4.3: Exemplificação da organização dos dados em *RDBMS* e bases de dados *NoSQL* orientadas a documentos[1]

Não obstante, apesar da adoção de uma base de dados NoSQL ajudar a complementar as limitações da *RDBMS* utilizada no projeto, é importante salvaguardar que a introdução de mais uma base de dados implica um aumento da complexidade de *deployment* do projeto, devendo este passo ser cuidadosamente ponderado[49]. Por esta razão, e atendendo ao âmbito do estágio, considerou-se que, neste momento, a adoção de uma base de dados NoSQL não seria uma opção viável, dado o seu impacto ao nível arquitetural e de *deployment*.

No entanto, salienta-se que, face ao aumento de popularidade que as bases de dados NoSQL têm vindo a registar nos últimos anos[2], as *RDBMS* têm procurado tornar-se mais flexíveis, introduzindo suporte nativo ao armazenamento de campos em formatos agnósticos como o *XML* e o *JSON*. Este tipo de prática promove uma maior versatilidade ao permitir a definição de dados semi-estruturados, complementando os tradicionais *schemas* fixos com campos especiais que permitem o armazenamento de documentos *schemaless*. De destacar igualmente a importância do suporte nativo para que estes campos possam ser: indexados, mapeados

[1]Referência para a imagem: https://content-static.upwork.com/blog/uploads/sites/3/2015/06/02170023/relational-vs-nonrelational-databases.png

[2]The rise of NoSQL databases: http://www.ibmbigdatahub.com/blog/rise-nosql-databases

de forma relacional e pesquisados, recorrendo-se a semânticas complementares ao *SQL* tais como o XPath[50] e o XQuery[51].

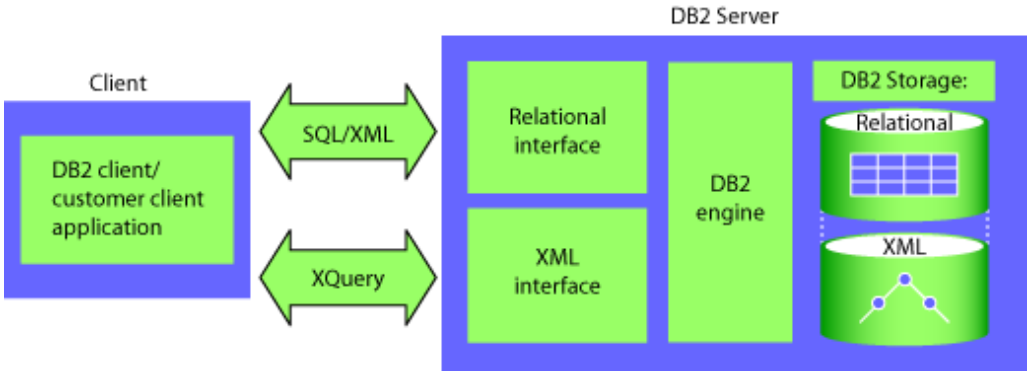


Figura 4.4: Vista geral do suporte nativo do DB2 a dados estruturados e não estruturados (XML)[52]

Nesta ordem de ideias, refere-se que a base de dados em utilização no projeto, o IBM DB2 9.7[1], disponibiliza um conjunto de mecanismos para armazenar e interagir com dados representados em *XML*[52][53][54], conforme pode ser observado na figura 4.4.

Assim, procurando tirar partido das potencialidades do DB2, acima referidas, foi possível definir um *schema* que beneficia tanto do modelo relacional como do não relacional, permitindo que seja feito o armazenamento de um relatório segundo uma estrutura orientada a documentos, conforme pode ser visto na figura 4.5.

Na figura 4.5 é possível observar o esquema genérico de dados, criado para suportar o armazenamento de qualquer relatório que, por razões de desempenho, não possa ser obtido de forma imediata, devendo ser computado periodicamente e persistido minimizando assim o impacto da sua geração em visualizações posteriores.

A entidade *RPT_PANEL* apresenta um conjunto de campos que permitem definir e caracterizar um relatório:

ID - Identificador textual único que identifica um relatório;

[1]Página oficial do IBM DB2: `http://www.ibm.com/analytics/us/en/technology/db2`

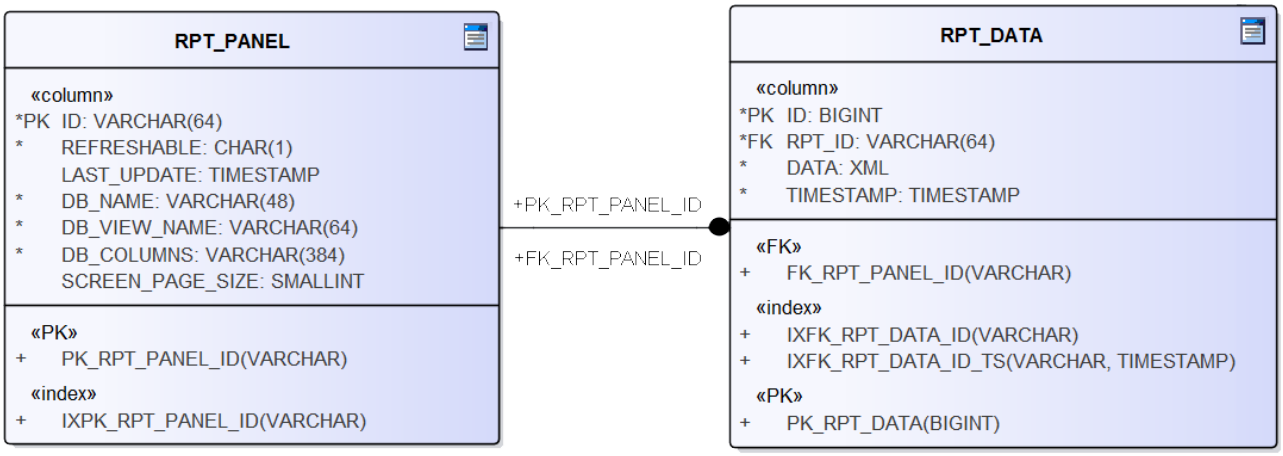


Figura 4.5: Modelo de dados do Módulo de Relatórios - Diagrama físico

REFRESHABLE - Carácter único que indica se o relatório deve apresentar um botão de atualização de dados, a nível da sua visualização;

LAST_UPDATE - Indicador temporal relativo à última atualização dos dados. Note-se que este campo é apenas utilizado para relatórios de obtenção diferida;

DB_NAME - Nome da base de dados de onde os dados são obtidos;

DB_VIEW_NAME - Nome da vista que permite obter os dados;

DB_COLUMNS - Nomes das colunas a serem exibidas no relatório;

SCREEN_PAGE_SIZE - Este campo opcional permite sobrepor a configuração genérica do número de linhas a serem exibidas num ecrã, sendo particularmente útil face à limitação da *framework* utilizada no *front-end* na criação de interfaces responsivas, conforme abordado na secção 4.8.

A entidade *RPT_DATA* tem como objetivo o suporte ao armazenamento de um relatório, podendo destacar-se os seguintes campos:

RPT_ID - Chave forasteira que identifica o tipo de relatório em causa;

DATA - Campo do tipo *XML* (nativo no DB2) que permite armazenar um relatório sob a forma de um documento daquele formato;

TIMESTAMP - Representa o instante em que os dados foram obtidos.

Este modelo de dados permite que seja definida uma estrutura comum a todos os relatórios, sendo complementada através de um campo *XML*, nativo à base de dados, que garante uma flexibilidade reforçada ao suportar dados heterogéneos, adaptando-se assim a qualquer necessidade futura ao nível do armazenamento de relatórios.

4.6.2 Análise e obtenção de dados

Conforme especificado no início da secção 4.6, é objetivo integrante desta tarefa a obtenção do contexto dos dados necessários para a geração dos relatórios. Atendendo a que estes dados já são obtidos pela atual estrutura informal de envio de relatórios, através de correio eletrónico (conforme indicado na secção 1.3.1), considerou-se, como pressuposto que a obtenção destes dados não seria da responsabilidade do estagiário dado o seu contexto limitado do esquema de dados em utilização no projeto. Como consequência, não foi previsto a nível de planeamento qualquer esforço para a realização desta tarefa. No entanto, dado o risco número quatro, referido na secção 4.4, o atraso da disponibilização das *queries* de obtenção dos dados pode ter efeitos negativos no planeamento efetuado. Assim, dado que foram apenas disponibilizadas seis das dez *queries* necessárias, e sendo urgente a rápida obtenção das restantes, foi solicitado o acesso aos *scripts* geradores dos relatórios, de forma a que fosse feita a sua engenharia reversa. Apesar desta tarefa ter sido concluída com sucesso e sem ajuda externa, foi necessário despender tempo adicional o que veio a afetar o planeamento inicial.

4.6.3 Rotinas para o pré-processamento de dados

Tal como evidenciado na figura 4.1 e na tabela 4.1, alguns relatórios, tais como: o de *Acessos x Digitalização x Impressão*, o de *Processos de Cheques Visados* e o de *Processos anulados*, requerem uma etapa de pré-processamento antes de poderem ser visualizados. De forma a fazer face a esta necessidade, foi necessário desenvolver

uma componente adicional, externa ao sistema, que foi concretizada sob a forma de uma *Java Standalone Application*, responsável pela obtenção e manipulação dos dados necessários para este tipo de relatórios. Desta forma, identificam-se duas necessidades distintas às quais esta componente deve dar resposta:

- Execução de múltiplas consultas às bases de dados do projeto;
- Obtenção de dados adicionais relativos a processos de negócio provenientes da *BPMS*, recorrendo à sua *API* de *REST*.

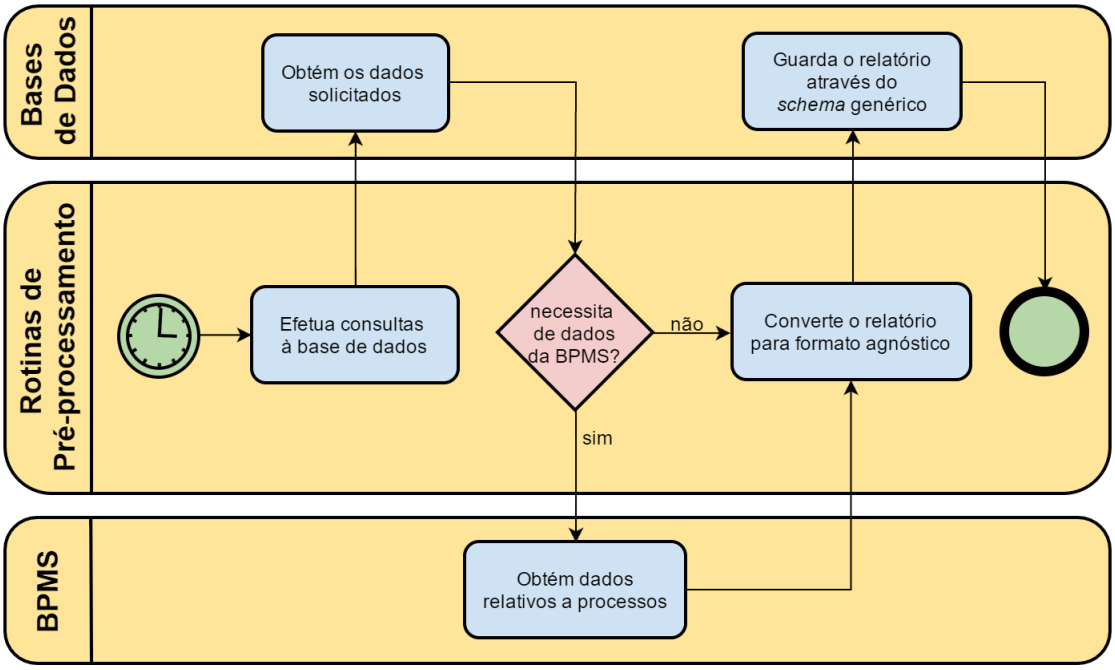


Figura 4.6: Diagrama *BPMN* representativo do funcionamento das rotinas

Na figura 4.6 é possível observar o ciclo de vida destas rotinas, bem como as principais componentes com a quais interagem. Cada rotina destina-se apenas a dar origem a um relatório, começando por efetuar um conjunto de consultas às bases de dados do projeto. Seguidamente, poderá ser necessário interagir com a *BPMS* a fim de obter informação adicional para relatórios que lidem, de forma direta ou indireta, com processos de negócio. No final, os dados que constituem um relatório são convertidos para *XML*, sendo armazenados através do recurso ao esquema de dados genérico descrito na secção 4.6.1.

Desenvolvimento

A conceção técnica de componentes de *software* reutilizáveis não constitui uma tarefa simples[55], podendo ser geralmente atribuída a fatores como a falta de contexto do problema ou do funcionamento do sistema, ou ainda à ausência de um domínio pleno das tecnologias visadas.

Atendendo às principais condicionantes identificadas na secção 4.3, consideramos que a reutilização de código é um fator desejável para colmatar as mesmas. Desta forma foi seguido o princípio *Refactoring To Generalize*, conforme proposto por Opdyke[55], que promove uma abordagem iterativa de *refactoring*[45] durante a fase de codificação. Esta prática, comum em ciclos de vida ágeis, como o *Extreme Programming*[38], contribuem para a redução de *code-smells*[56], potenciais indicadores de más práticas ou lacunas no desenho do sistema, a nível do código.

No apêndice D são ilustradas as principais classes que compõem as rotinas desenvolvidas. No primeiro item deste apêndice, destaca-se a classe *GenericReport* que tem como objetivo abstrair a criação de relatórios que necessitem de efetuar consultas à base de dados do *eMudar*. Por outro lado, realça-se um segundo nível de abstração, conseguido através da classe abstrata *GenericReportWithRest*, que estende a anterior, permitindo assim a criação de relatórios que necessitam de interação com a *BPMS*.

Sublinhe-se que as referidas abstrações constituem uma aplicação do *Template Method*, um *design-pattern* conhecido por promover a reutilização de código[4].

Integração com a *BPMS*

Conforme referido anteriormente, a interação com este sistema é feita através de uma *API* de *REST* existente para o efeito. Não obstante, é necessário proceder à análise e extração dos dados das resposta que se encontram em formato *XML*.

Durante a implementação deste tipo de relatórios, observou-se que obtenção dos campos relevantes recorrendo a expressões XPath[50] levou à existência de uma quantidade desadequada de código semelhante e de difícil legibilidade. No sentido

de combater esta situação, recorreu-se à implementação de uma anotação[1] facilitadora da implementação deste tipo de relatórios, reduzindo substancialmente a percentagem de código duplicado e simplificando o processo.

```
@XmlAttribute
@ReportField("//data/resultMap//item[@key=\"nomeTipoProcesso\"]/value//text()")
protected String instanceName;
```

Figura 4.7: Exemplo da utilização da anotação personalizada *ReportField* para a obtenção de um atributo de processo

A anotação *ReportField* exemplificada na figura 4.7 é utilizada na implementação de relatórios que requerem integração com a *BPMS*, permitindo a definição clara de um expressão XPath[50] necessária para a obtenção de um determinado campo. Considera-se que este mecanismo se revelou útil para reduzir a complexidade da implementação e aumentar a legibilidade do código, permitindo uma maior modificabilidade e extensibilidade do mesmo.

Por outro lado, recorreu-se também à anotação *XMLAttribute* possibilitando que o campo possa ser convertido para *XML* através da biblioteca JAXB[2] facilitando assim a sua persistência e utilizando o *schema* genérico especificado na secção 4.6.1.

Resiliência a falhas transitórias

Dado que estas rotinas interagem com duas componentes externas que podem estar sujeitas a falhas transitórias, foi aplicado o *Retry Pattern*[47]. A utilização deste *design-pattern* permite melhorar a estabilidade e robustez de uma aplicação, ao lidar de forma transparente com pontos sensíveis e potencialmente sujeitos àquele tipo de falhas. Entre estes, pode destacar-se a criação de ligações e comunicação entre serviços e a persistência de dados.

[1] JSR 250: Common Annotations for the JavaTM Platform: https://www.jcp.org/en/jsr/detail?id=250

[2] Página do Java Architecture for XML Binding: https://jaxb.java.net

4.6.4 Agendamento da atualização dos dados

De forma a possibilitar a atualização dos *relatórios de obtenção periódica*, conforme descrito na secção 4.2, recorreu-se à utilização do mecanismo *CRON*[1] para proceder ao seu agendamento. Este, consiste num serviço presente em sistemas UNIX que permite agendar a execução de comandos de forma simples e flexível. Saliente-se que esta solução possibilita, não só, a execução das rotinas, como também o despoletar da atualização das vistas materializadas (secção 4.6.1), respeitando ambas as restrições identificadas na secção 4.3. De destacar também que este mecanismo permite suportar tanto os ambientes de teste do projeto, assentes em *Red Hat Enterprise Linux*, como os de produção, assentes em AIX, uma distribuição proprietária da IBM baseada em UNIX.

4.6.5 Serviço de Relatórios

Respeitando a *Arquitetura de Serviços* na qual assenta o projeto *eMudar*, foi criado um serviço com vista a isolar a lógica de negócio associada ao módulo de relatórios.

Tal como os restantes *web services* em utilização no *eMudar*, o *Serviço de Relatórios* foi implementado através da tecnologia *SOAP*, sendo responsável pela obtenção dos dados relativos a relatórios, permitindo filtragem, paginação e ordenamento de registos, procurando tirar pleno partido das potencialidades da *RDBMS* para esse efeito. Apesar de neste momento estar apenas prevista uma utilização do serviço, de forma interna ao projeto, seria imprudente descuidar a sua segurança, sobretudo considerando que interage diretamente com a base de dados. Desta forma, foram seguidas duas abordagens destinadas a garantir um nível superior de segurança:

Utilização de *Prepared Statements* - Esta prática permite aumentar a segurança e o desempenho de determinados procedimentos à base de dados, protegendo-a contra ataques do tipo *SQL Injection*, considerada pela *OWASP* como a vulnerabilidade mais grave em aplicações *web*[57];

[1] Manual do serviço CRON: `https://help.ubuntu.com/community/CronHowto`

Validação de todos os parâmetros de entrada - De forma a evitar comportamentos inusitados por parte do serviço, face a parâmetros inválidos e/ou maliciosos, deve ser feita uma validação integral de todos os parâmetros de entrada. No sentido de validar a correta implementação deste mecanismo, o serviço foi sujeito a testes de robustez conforme detalhado na secção 5.5.3.

4.6.6 Camada de Apresentação

O desenvolvimento desta camada foi feito recorrendo à *framework ZK*, em utilização no projeto, que respeita a filosofia *MVC*. Este *design-pattern* garante a definição de três camadas de diferentes deveres, resultando numa eficaz separação de responsabilidades e promovendo uma modificabilidade superior[4][5][45].

Durante a implementação, optou-se por utilizar a vertente *MVP* (uma variante do *MVC*), suportada pela *framework*, de forma a atingir um menor acoplamento entre as diversas camadas, ao impor que o *Presenter* faça a mediação da comunicação entre a *View* e o *Model*[5], exemplificado na figura 4.8.

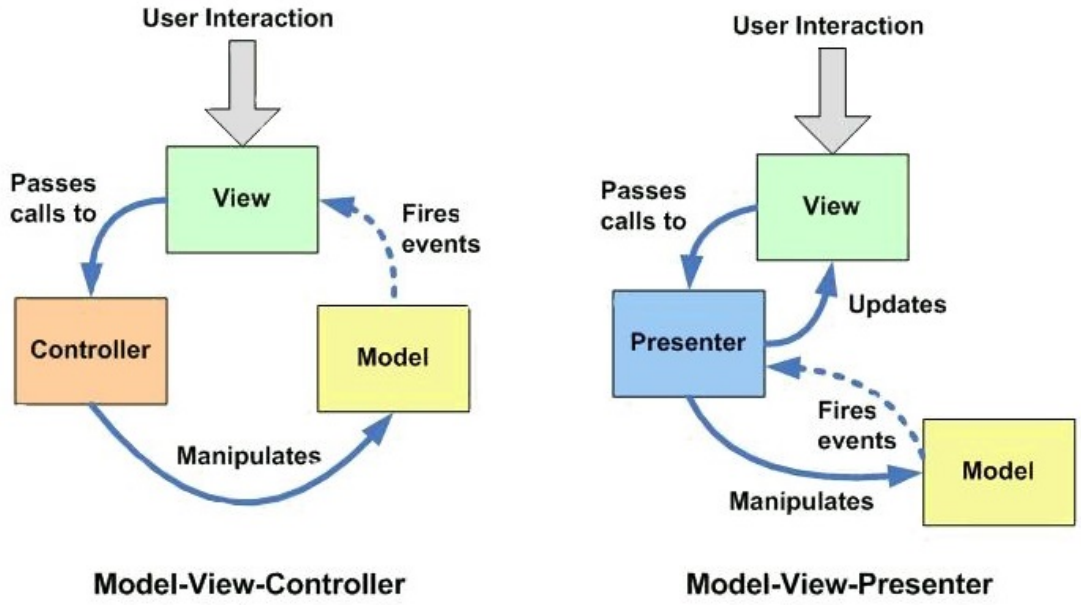


Figura 4.8: Comparação entre o modelo *MVC* e a sua variante *MVP*[1]

[1]Referência da imagem: `http://assets.devx.com/articlefigs/18289.jpg`

Na arquitetura do módulo, representada na figura 4.2, é possível observar este mesmo comportamento, sendo que apenas os *Presenters* comunicam com os *Models*, representados pelos *Web Services* e pela *BPMS*.

Relembrando a figura 4.1, foi necessário desenvolver quatro conjuntos de *presenter/view*, a saber:

GenericReportPresenter - Esta componente serve todos os relatórios, sendo instanciada pelos relatórios genéricos, de acordo com a secção 4.2. Para além disso, constitui uma base que deve ser extensível para lidar com as particularidades de relatórios que necessitem de uma visualização mais personalizada;

AdminSysProcessesReportPresenter - Estende a componente anterior, dando suporte ao ecrã de *Processos atribuídos ao Administrador de Sistema*;

BranchReportPresenter - Dá suporte ao ecrã de *Acessos x Digitalização x Impressão*, constituindo uma simples extensão do *GenericReportPresenter*;

AdminSysProcessDetailPresenter - Dá suporte ao ecrã de *Detalhe do Erro* que, não constituindo propriamente um relatório, permite uma visualização mais detalhada da instância de um processo de negócio, no âmbito do ecrã de *Processos Atribuídos ao Administrador de Sistema*. Por esse motivo não constitui uma extensão da interface genérica.

4.7 Valências

Nesta secção enunciam-se alguns dos principais pontos fortes que se destacam relativamente ao desenvolvimento do Módulo de Relatórios.

Conformidade com *design-patterns*

No decorrer da fase de desenvolvimento foram identificados um conjunto de aspetos que constituem problemas de engenharia habituais na conceção e implementação de *software*. Desta forma, optou-se por seguir um conjunto de *design-patterns* que constituem soluções gerais e bem estudadas para problemas recorrentes em

determinados contextos. Saliente-se, que este tipo de práticas contribui para a criação de soluções mais compreensíveis e de maior qualidade na utilização deste tipo de estratégias.

Robustez

Durante os trabalhos realizados no âmbito do Módulo de Relatórios, houve uma preocupação constante no que diz respeito à validação de parâmetros de entrada no sistema e ao tratamento conveniente de exceções, ainda que este tipo de parâmetros não se encontrasse definido na sua especificação de requisitos. Desta forma, foi necessário proceder à recolha da gama de valores aceitáveis para cada parâmetro de entrada, de forma a proceder à sua validação.

A nível da interface gráfica, a *framework ZK* permitiu que fosse feita uma validação completa, do lado do cliente, recorrendo a expressões regulares avaliadas através de *javascript*, impedindo assim a realização de pedidos inválidos ao servidor.

Neste sentido, são detalhados na secção 5.5 os testes de robustez efetuados para comprovar esta propriedade da implementação elaborada.

Documentação

Relembrando que a fase de manutenção constitui a maior parte dos custos de um *software*[44], é essencial que sejam tomadas precauções para manter esse custo tão reduzido quanto possível. Assim, considera-se essencial a existência de documentação que facilite a compreensão simples do funcionamento do módulo, tanto ao nível do código, como do esquema de dados. Nesse contexto, recorreu-se:

- Ao *Javadoc*[1] para descrever o código desenvolvido;
- A comentários a nível do código para explicar aspetos mais complexos;
- A descrições de entidades e de colunas constituintes do modelo de dados, tirando partido das potencialidades de documentação da *RDBMS*.

[1]Página da Oracle sobre o Javadoc: `http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html`

Refactoring de Código

Conforme mencionado na secção 4.6.3, durante toda a fase de codificação, o código fonte foi sujeito a múltiplos *refactorings*, prática comum em ciclos de vida ágeis, como o *Extreme Programming*[38], contribuindo para a redução de *code-smells*[56], potenciais indicadores de más práticas ou lacunas no desenho do sistema, a nível do código.

Logging

A utilização de boas práticas de *logging* no desenvolvimento de *software* é importante para promover uma manutenibilidade superior, contribuindo para o registo de informações que auxiliem em tarefas como o diagnóstico de comportamentos inusitados bem como a análise de desempenho e de auditoria.

De destacar que as bibliotecas modernas de *logging* são facilmente configuráveis por ficheiro, possibilitando a definição de diferentes níveis de granularidade nos seus registos, consoante o módulo ou componente visado. Permitem, também, que o formato dos registos possa ser definido transversalmente em todo o projeto, facilitando a sua indexação por ferramentas específicas para o efeito.

Assim, e de forma a evitar o *lock-in* face a uma ferramenta específica, optou-se por utilizar o *Simple Logging Facade for Java*[1] (SLF4J), uma *façade*[4] que disponibiliza uma interface simplificada que funciona como uma abstração à sintaxe utilizada. Esta, é suportada por inúmeras implementações de bibliotecas de *logging*, ficando essa escolha ao critério do programador, podendo ser prontamente substituída sem necessidade de qualquer alteração a nível de código.

4.8 Desafios e Dificuldades

Muitos dos principais obstáculos que surgiram ao longo do desenvolvimento do Módulo de Relatórios foram previamente identificados, tanto a nível das condicionantes do estágio (secção 1.5), como a nível da Análise de Riscos desta componente (secção 4.4).

[1]Página oficial do SLF4J: `http://www.slf4j.org`

De salientar, porém, que o limitado contacto prévio do projeto *eMudar*, aliado à ainda reduzida experiência do estagiário, constituíram fatores desafiantes para a concretização do estágio sem prejuízo da qualidade do trabalho ou do calendário proposto. Neste sentido, destaca-se mais uma vez a complexidade do projeto, que conta com cerca de um milhão de linhas de código e evidencia uma forte componente de integração ao interagir com múltiplos sistemas e serviços externos. Também a utilização de *software* e de bibliotecas obsoletas no projeto contribui para uma dificuldade acrescida das tarefas de implementação, apresentando limitações a nível de funcionalidade e de documentação, resultando num desenvolvimento mais verboso e moroso. Como exemplo deste facto, listam-se as seguintes tecnologias em utilização no projeto, sendo indicada a data de surgimento e de final de suporte: Java 6 (2006-2013)[1], Java EE 5 (2006)[2], ZK 5 (2010-2013)[3], IBM DB2 9.7 (2009)[4]. Realça-se ainda o exemplo da biblioteca *ZK*, cuja versão em utilização no projeto não permite a elaboração de páginas *web* através de um *design* responsivo[58], constituindo um entrave à conceção de ecrãs que funcionem independentemente da resolução do ecrã utilizado. Para avaliar o impacto desta limitação, atendendo à diversidade de resoluções de ecrã em utilização pelo *BFA*, optou-se por cobrir este ponto nos testes de robustez à interface gráfica do módulo, conforme detalhado na secção 5.5.2.

Para além das situações acima referidas, também se registaram outro tipo de contratempos, de carácter imprevisível, que não podiam ser previstos inicialmente. A saber:

- Escassa e desatualizada documentação técnica do projeto;
- Limitações de completude a nível dos requisitos do módulo, o que obrigou a uma revisão e clarificação dos mesmos, conforme descrito na secção 5.2.

[1]Java Version History: `https://en.wikipedia.org/wiki/Java_version_history`
[2]Java EE version history: `https://en.wikipedia.org/wiki/Java_EE_version_history`
[3]ZK Framework Release Note List: `https://www.zkoss.org/product/zk/releaseNote`
[4]IBM DB2 - Wikipedia - `https://en.wikipedia.org/wiki/IBM_DB2`

4.9 Trabalho Futuro

Identificação de pontos adicionais de extensibilidade

Atendendo às principais condicionantes identificadas para o presente módulo (secção 4.3), considera-se que seria interessante, numa fase posterior, reanalisar as necessidades do cliente, com vista à identificação de eventuais novos "pontos de extensibilidade". Saliente-se que este tipo de ações são importantes para uma correta manutenção dos atributos de qualidade de um *software*.

Complemento de Testes Unitários

De forma a que se possam aplicar, com sucesso, técnicas como o *refactoring* de código, conforme mencionado na secção 4.7, é essencial que exista uma elevada cobertura do código a nível de testes[56]. Tal facto, deve-se à confiança adicional que os testes trazem, ao contribuir para uma deteção imediata de eventuais defeitos introduzidos durante esta operação de reestruturação. Neste sentido, considera-se que seria vantajosa a criação de testes unitários para a aplicação responsável pelo enriquecimento dos dados que constituem os relatórios.

Otimização da obtenção do ecrã de Processos Atribuídos ao SysAdmin

Este relatório de obtenção imediata (ver figura 4.1), faz uso de uma complexa vista de base de dados que pode demorar até dez segundos a ser gerada. De acordo os estudos efetuados por Nielsen[59], relativamente ao impacto da latência na experiência de utilização, verifica-se que o período de obtenção deste relatório se encontra no limite de atenção de um utilizador, podendo assim, afetar a eficácia da sua utilização. Ressalta-se que a referida vista de base de dados não é específica para este ecrã, apresentando alguns campos desnecessários cuja obtenção constitui uma perda de eficiência. Durante o estágio, considerou-se a construção de uma vista dedicada alternativa, contando apenas com os campos necessários. No entanto, atendendo ao conhecimento limitado do *schema* da base de dados, por parte do estagiário, e a constrangimentos de ordem temporal, não foi possível explorar esta hipótese, sobretudo na ausência de uma garantia que o ganho de desempenho fosse significativo.

4.10 Conclusões

Durante a implementação do Módulo de Relatórios foi possível contactar com diversas etapas do *SDLC*, em contexto real e aplicadas a um sistema crítico, onde o rigor e a qualidade são atributos totalmente indispensáveis. Apesar do trabalho se ter revelado mais complexo do que o inicialmente previsto, verificou-se que a cuidadosa análise de riscos e a consequente definição de planos de mitigação, foram eficazes, contribuindo para que todos os objetivos fossem atingidos com sucesso.

Destacam-se, também, um conjunto de aspetos relevantes identificados ao longo do estágio, e que se consideram dignos de menção:

Estimativas

De acordo com Robert Glass[44], um dos principais fatores que contribuem para o descontrolo de um projeto de *software* é o otimismo nas estimativas. Este fator foi amplamente considerado durante o estágio, tendo sido endereçado a nível da análise de riscos e mitigado recorrendo a técnicas de estimação conforme descrito na secção 3.2.4. Ainda assim, ocorreram derrapagens a nível do planeamento, não devido às estimativas em si, mas à ausência da identificação de algumas tarefas, tais como: análise dos mecanismos de obtenção de dados para os relatórios, elaboração de testes unitários e fase de sincronização de código e estabilização. De destacar, porém, que foi positiva a antecipação, desde o início, de dois dias para a correção de defeitos, contribuindo para atenuar o impacto existente no planeamento. Concluiu-se, assim, que a alocação de uma margem temporal de segurança constitui uma boa prática para mitigar a eventual ocorrência de contratempos num projeto.

Integração Contínua e Qualidade de Código

Observou-se que, na presença de um elevado número de pessoas a fazer alterações de código, é fundamental que existam mecanismos eficazes de integração contínua, que garantam o sucesso das etapas de compilação e de execução dos testes unitários e de sistema. Porém, para assegurar a eficácia destas operações, é fundamental que tanto a compilação, como a execução dos testes sejam operações

pouco demoradas. Para tal, é por vezes necessária a definição de diferentes níveis de testes, possibilitando a execução frequente de testes rápidos, deixando os testes mais complexos e lentos para fora da hora de expediente. Por outro lado, considera-se que deveriam ser tomadas precauções a fim de assegurar uma melhor gestão do ramo principal de código (*trunk*), obrigando a que a incorporação de *feature-branches* (ramos de código específicos para uma nova funcionalidade) fosse precedida pela validação do sucesso da compilação, alvo de testes automáticos, análise estática, formatação de código e, opcionalmente, uma inspeção manual de código. Considera-se que estes mecanismos e práticas, quando bem aplicados, contribuem para um desenvolvimento mais célere e para a deteção antecipada de eventuais defeitos.

Arquitetura do *eMudar*

A *Arquitetura Orientada a Serviços* (*SOA*) do *eMudar* continua a dar resposta às necessidades do BFA, evidenciando um satisfatório grau de extensibilidade. No entanto, na ausência de uma revisão à arquitetura e de estratégias habituais em ciclos de vida mais ágeis como o *refactoring*[56], começa-se a evidenciar uma certa erosão arquitetural[60], resultando num maior acoplamento entre as componentes do sistema. Como consequência, destaca-se: a amplificação de falhas, redução da modificabilidade e limitações a nível da extensibilidade, afetando a manutenibilidade do sistema, resultando num desenvolvimento mais dispendioso. Não sendo este fenómeno inédito, salienta-se que as arquiteturas monolíticas, como o caso do *eMudar*[61], apresentam uma certa propensão a este tipo de efeitos[62].

Plano de atualização de bibliotecas em final de vida

Apesar de ser necessário investir algum esforço na concretização desta tarefa, englobando a elaboração de um planeamento, análise de impacto, fase de experimentação e testes; considera-se que, a médio prazo, esta medida traria efeitos benéficos para o projeto, contribuindo para um desenvolvimento mais eficiente. Para fundamentar esta afirmação, destaca-se que a constante implementação de soluções para colmatar limitações das tecnologias, provoca um impacto negativo a nível de manutenibilidade do projeto. Ainda que eficazes a curto prazo, este tipo de

práticas contribui para um aumento da "dívida técnica", situação que não pode continuar indefinidamente a médio/longo prazo[56].

Criação de documentação técnica

Dadas as limitações a nível da documentação técnica referidas na secção 4.8, considera-se seria de extrema utilidade a criação de um Guia de Anotações e um Guia de reutilização de componentes de Interface Gráfica, com vista a promover a correta utilização destes mecanismos e consequente redução de código duplicado.

Capítulo 5

Garantia de Qualidade - Módulo de Relatórios

Neste capítulo é abordado todo o trabalho desenvolvido no âmbito da Garantia de Qualidade do Módulo de Relatórios do *eMudar*.

De acordo com o *SDLC* em vigor no projeto, descrito na secção 3.2.1, a fase de Garantia de Qualidade assume um papel fulcral na conceção de qualquer módulo do sistema *eMudar*, assegurando que é feita uma verificação integral das várias etapas que constituem o seu desenvolvimento.

Assim, começou-se por analisar os principais riscos que podem afetar o sucesso desta etapa do estágio, sendo propostas um conjunto de medidas para mitigar a sua eventual ocorrência. De seguida, procede-se à inspeção da Especificação de Requisitos do módulo, onde são identificados diversos aspetos a melhorar, com vista à obtenção de uma usabilidade e manutenibilidade superiores. Feito isto, os requisitos revistos foram utilizados para a conceção de uma Especificação de Casos de Teste, parte integrante do Plano de Testes do módulo (ver apêndice E), onde é descrita a abordagem, a descrição das componentes a testar, o mapeamento entre testes e requisitos, as necessidades do ambiente, entre outros. Seguidamente, são descritos os esforços efetuados na criação de uma suite de testes de aceitação automáticos, enquanto estratégia mitigadora do risco número um desta etapa.

Entretanto, são referidos os Testes de Robustez efetuados, garantindo uma verificação complementar aos testes de aceitação previamente descritos. Por último, mas não menos importante, são mencionados alguns aspetos relevantes a realizar em trabalhos futuros, seguido das conclusões e considerações finais desta etapa do estágio.

5.1 Análise de Riscos

No âmbito da fase de Garantia de Qualidade do estágio e com base no processo de Análise de Riscos especificado na secção 3.2.2, podemos enumerar um conjunto de riscos que foram considerados mais relevantes:

1. Testar o sistema *eMudar* por completo requer um elevado esforço manual, pelo que nem sempre todas as componentes são verificadas. Redução da confiabilidade ao lançar novas versões, contribuindo para a redução da qualidade e aumento dos custos a médio prazo;
2. O módulo de relatórios não se encontra implementado, estando portanto sujeito a alterações de requisitos. O caderno e plano de testes poderão ter de ser revistos, sendo necessário tempo adicional para revisões;
3. O estagiário possui reduzida experiência na área de garantia de qualidade. As estimativas poderão ser desajustadas dada a potencial necessidade de revisão do trabalho, o que pode comprometer o sucesso do estágio;
4. Uma boa parte do projeto assenta numa suite proprietária e fechada de *BPM*, apresentando limitações a nível de integração e impondo um determinado fluxo de tarefas. Dificuldades acrescidas em tornar os testes idempotentes, o que complica a automatização dos mesmos.

Tabela 5.1: Matriz de Probabilidade-Impacto de riscos identificados na componente de *SPAE*

Probabilidade	Impacto		
	Marginal	Crítico	Catastrófico
Alta		2,4	
Média		3	1
Baixa			

Mitigação

1. Considerar a automatização dos testes, contribuindo para uma maior confiabilidade no módulo após a sua implementação -> Implica que seja considerada, no planeamento, uma tarefa específica para o efeito. Assim sendo, deverá ser organizado um *workshop* em automação de testes por um dos elementos mais experiente da equipa, durante a fase de análise, de forma a reduzir a curva de aprendizagem associada;

2. Deve ser feita uma revisão dos requisitos durante a fase de análise, permitindo que sejam solicitadas eventuais alterações de forma antecipada, precedendo assim a escrita da Especificação de Casos Teste e a implementação do módulo. Deve ser mantido um mapeamento direto entre testes e requisitos, de forma a garantir a sua rastreabilidade, facilitando assim a execução de alterações no futuro;

3. Deve ser organizada uma sessão de tutoria, durante o 1º semestre, sobre o processo de Garantia de Qualidade que vigora no projeto. Esta sessão será assegurada por um elemento sénior da equipa e deverá preceder as atividades do estágio no âmbito da garantia de qualidade;

4. Deve utilizar-se uma réplica da base de dados do *BPM* para garantir um meio estável e controlado, minimizando o impacto da ausência de idempotência de alguns testes. Como objetivo *nice-to-have*, poder-se-á estender a suite de testes, automatizando a injeção de processos de teste, tornando-a assim totalmente idempotente.

5.2 Revisão de Requisitos

De acordo com a estratégia de mitigação definida para o risco número dois desta componente do estágio, optou-se por fazer uma inspeção à Especificação de Requisitos do Módulo de Relatório (apêndice B). Considera-se que esta prática promove a deteção antecipada de eventuais defeitos que pudessem surgir durante ou após as fases de conceção de testes e implementação, resultando em custos adicionais indesejáveis.

A inspeção efetuada permitiu a deteção e correção de alguns aspetos a melhorar, sendo resumidos nas seguintes categorias:

Desempenho
Observou-se que o ecrã de *Acessos vs Digitalização vs Impressão* apresentava um peso computacional demasiado elevado para poder ser obtido em tempo real passando em alternativa, a ser obtido de forma diferida, com periodicidade configurável de acordo com as necessidades da instituição;

Completude
Um requisito deve ser tão completo e explícito quanto possível, evitando ambiguidades e a possibilidade de múltiplas interpretações. Neste sentido, foram detetados alguns tópicos que necessitaram de ser expandidos de forma a prevenir potenciais dúvidas em fases posteriores do *SDLC*. Como exemplo, destacamos:

- A omissão da periodicidade de renovação do ecrã de *Acessos vs Digitalização vs Impressão*;
- A ausência da definição da nomenclatura para o ficheiro criado no âmbito da funcionalidade de exportação de um relatório para Excel.

Usabilidade
De acordo com as heurísticas propostas por Nielsen[59], observaram-se alguns defeitos a nível de consistência e de modelo mental:

- **Marcas Temporais** - Inicialmente, apenas os ecrãs que apresentavam dados obtidos em tempo real, possuíam uma marca temporal indicadora da data de obtenção dos dados. No entanto, considera-se que a ausência desta funcionalidade nos relatórios de obtenção periódica, constituiria um defeito de usabilidade, impedindo que os utilizadores tivessem a noção clara do *time-frame* dos dados. Para evitar esta situação, aplicou-se a marca temporal a todos os relatórios, proporcionando uma maior consistência dentro do módulo;
- **Botão de atualização de dados** - Uma vez que este botão apenas pode ser utilizado no âmbito de relatórios de obtenção imediata, considera-se que o mesmo não deve estar disponível nos relatórios de obtenção periódica diferida, de forma a evitar confusões por parte dos utilizadores.

Nomenclatura

Por fim, foram ainda feitas algumas revisões a nível da nomenclatura utilizada no Módulo de Relatórios, a fim de proprocionar um registo escrito mais formal e apropriado para os relatórios em questão. Destacam-se, assim, as seguintes categorias:

- **Remoção de anglicismos desnecessários** - E.g. Processos *Failed*, *Report* SMSs, entre outros;
- **Utilização de nomes mais claros** - E.g. **Lista** de respostas corretas negativas em vez de **Número** de respostas corretas negativas, uma vez que este tipo de relatórios consiste numa listagem de elementos e não num número como o título dava a entender.

5.3 Testes de Aceitação

Entende-se por teste de aceitação, todo e qualquer teste de carácter formal, associado às necessidades do utilizador, requisitos e processos de negócio, realizado com o objetivo de determinar se existe ou não um sistema que satisfaça os critérios de

aceitação, permitindo ao utilizador, cliente ou entidade autorizada, determinar se deve ou não aceitar o sistema [IEEE 610][41].

O *SDLC* em vigor no projeto prevê a entrega de um caderno de testes de aceitação juntamente com a disponibilização de cada módulo. Assim, atendendo a que estes testes dependem apenas da especificação de requisitos, a sua elaboração começa logo após a aprovação dos requisitos por parte do Cliente, precedendo, por norma, a fase de desenvolvimento.

O Plano de Testes que inclui a especificação de casos de teste elaborada no âmbito desta secção podem ser consultados no apêndice E.

5.4 Automatização de Testes de Aceitação

O **principal objetivo** desta etapa do estágio é a conceção de uma **prova de conceito** em automatização de testes, aplicada ao Módulo de Relatórios do *eMudar*. Esta, deve permitir efetuar a verificação dos requisitos do referido módulo, através da execução automática dos testes de aceitação previamente definidos.

Pretende-se também promover o desenvolvimento de espírito crítico nesta vertente, de garantia de qualidade de *software*, avaliando a viabilidade da adoção deste tipo de técnica num projeto de elevada complexidade como o *eMudar*.

5.4.1 Motivação e Objetivos da Automação

Quando corretamente aplicada, esta técnica contribui para o aumento da qualidade de um *software*, ao promover uma execução mais rápida e frequente dos testes[63]. Este tipo de práticas favorece uma redução do risco operacional, ao propiciar uma deteção precoce de defeitos[36], constituindo assim uma poupança a médio/longo prazo[63].

Entre os principais **objetivos**, destacam-se os seguintes:

- Obtenção de um melhor entendimento, compreensão e confiança no Sistema em Teste (*SUT*), favorecendo a criação de *releases* mais frequentes;
- Validação automática dos requisitos exercitados nos testes de aceitação;
- Deteção precoce de defeitos, fornecendo uma sequência de passos para a reprodução do problema;
- De fácil execução, podendo ser despoletados por mecanismos de integração contínua;
- De elevada manutenibilidade e extensibilidade, constituindo uma economia de escala no futuro.

5.4.2 Arquitetura de Testes

Na figura 5.1 é possível observar uma ilustração da arquitetura utilizada no âmbito dos testes automatizados ao Módulo de Relatórios, evidenciando-se as suas principais componentes.

No nível mais inferior, encontra-se representada a lógica dos testes, assim como os dados que são utilizados como suporte, respeitando a filosofia *data-driven*, conforme descrito na secção 2.3. Estes, são executados recorrendo à *Robot Framework*, a ferramenta de automatização de testes eleita, que pode utilizar diversas bibliotecas para interagir com o sistema em teste. Dado que o *eMudar* está assente em tecnologias *web*, foi utilizada a biblioteca *Selenium*[1] para fazer a interação com o *browser*, simulando a interação de um utilizador real com o sistema. Destaca-se ainda a flexibilidade que a *Robot Framework* apresenta, disponibilizando um conjunto alargado de bibliotecas de teste, a utilizar consoante o tipo de *SUT*, permitindo também que sejam implementadas bibliotecas adicionais para endereçar necessidades específicas. Assim, foi desenvolvida a biblioteca *User Profile Resolver*, recorrendo à linguagem de programação *Python*, de forma a colmatar um dos desafios identificados na secção 5.7.2, relacionado com a divergência de utilizadores e perfis entre ambientes. Esta biblioteca interage com a base de dados do

[1]Página oficial da ferramenta *Selenium*: `http://www.seleniumhq.org`

módulo de *Organização e Autorização*, obtendo os utilizadores que possuem acesso às funcionalidades visadas nos testes, para um determinado ambiente. De referir, que este mecanismo se encontra descrito com maior detalhe no parágrafo sobre *Abstração de perfis, utilizadores e ambientes* da secção 5.6.

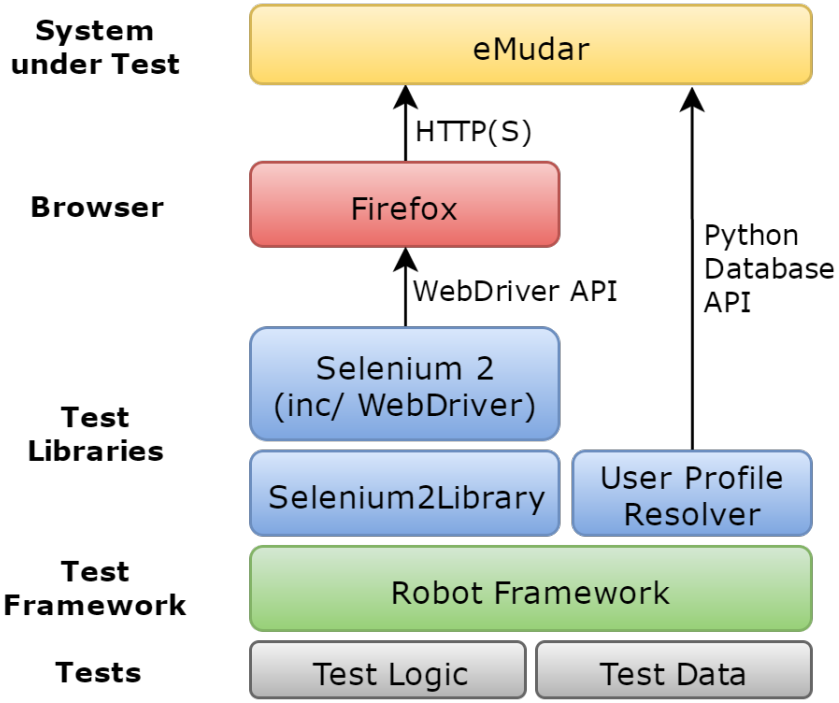


Figura 5.1: Arquitetura dos Testes de Aceitação Automáticos[1]

5.4.3 Fiabilidade da Suite de Testes

Entende-se por fiabilidade a capacidade de um *software* de executar um conjunto de funções necessárias, sob determinadas condições, durante um determinado período de tempo ou número de execuções [ISO 9126][41].

Relembrando que a suite de testes implementada representa uma prova de conceito para a utilização de testes automáticos no projeto *eMudar*, torna-se fulcral assegurar a fiabilidade e a robustez da mesma, demonstrando a viabilidade da abordagem seguida. Para tal é indispensável provar que os testes implementados

[1]Baseado na imagem: http://robotframework.org/img/architecture-big.png

não são frágeis (um *anti-pattern* em automação de testes[36]), garantindo assim um dos principais objetivos em automação de testes, a saber: "Um teste só deve falhar perante defeitos existentes na implementação do sistema"[63].

Nesse sentido, foi elaborado um *script* que executasse a bateria de testes de forma sucessiva, recolhendo métricas acerca do tempo despendido e do número de falhas registado. No total, efetuaram-se quatro ensaios constituídos por pelo menos trinta execuções sucessivas da suite. Entre elas, procedeu-se a uma análise em detalhe dos relatórios gerados pelos testes em falha, tendo as conclusões sido registadas na tabela 5.2.

Aquelas falhas foram categorizadas segundo quatro categorias distintas, identificando diferentes níveis de responsabilidade na sua correção ou mitigação:

Falhas na Suite de Testes - Correspondem a defeitos associados à implementação da Suite que foram corrigidos no âmbito da automatização de testes;

Defeitos de Implementação - Correspondem a falhas resultantes de defeitos na implementação do módulo de relatórios, que foram corrigidos no âmbito do seu desenvolvimento;

Falhas inesperadas de sistema - Correspondem a falhas causadas por comportamentos inesperados por parte do sistema. Apesar de não poderem ser inteiramente "corrigidas", deve-se procurar minimizar a sua ocorrência, através da identificação de fluxos de execução que sejam suscetíveis a situações deste tipo, através da implementação de mecanismos de tolerância. A título de exemplo, destaca-se que práticas como a reexecução de tarefas, se mostraram eficazes para lidar com falhas transitórias;

Perda de Sessão - Apesar de serem falhas inesperadas do sistema, optou-se por criar uma categoria específica para estas situações dado que apresentaram uma expressão relevante.

Na tabela 5.2 observa-se a evolução da fiabilidade da suite de testes automáticos entre ensaios, sendo também evidenciado o esforço dedicado à sua estabilização. Destaca-se que entre ensaios foram desencadeadas ações corretivas e preventivas,

Tabela 5.2: Evolução da Fiabilidade da Suite de testes automáticos

Estatísticas	**1º Ensaio**	**2º Ensaio**	**3º Ensaio**	**4º Ensaio**
Nº Execuções	42	39	46	55
Taxa de Sucesso	71%	87%	93%	98%
Tempo Médio (mm:ss)	10:30	10:46	11:51	12:11
Desvio Padrão (mm:ss)	01:02	00:56	00:19	00:35
Tipo de Falha				
Suite de Testes	1 (8%)	1 (20%)	0	0
Defeito de Implementação	7 (58%)	1 (20%)	0	0
Inesperadas de Sistema	0	2 (40%)	3 (100%)	0
Perda de Sessão	4 (33%)	1 (20%)	0	1 (100%)

de forma a corrigir e a mitigar as falhas detetadas, o que contribui para um aumento da estabilidade e robustez da suite de testes.

5.4.4 Defeitos de Implementação identificados

Nesta secção são apresentados os defeitos a nível da implementação do Módulo de Relatórios que foram identificados através da suite automática de testes. Optou-se por apresentá-los sob a forma de categorias de forma a fornecer uma visão mais sucinta dos mesmos.

Exceções não consideradas

Nem sempre é evidente a identificação de todas as exceções que um determinado serviço pode causar, podendo essa falta de clareza resultar em comportamentos inusitados. A título de exemplo destacam-se os pedidos aos *web-services* que para além das exceções às quais estão explicitamente sujeitos, podem ainda ser afetados por *timeouts* e outras exceções inerentes ao protocolo usado. A sua ocorrência pode levar a que no limite um ecrã não seja exibido devido à falha da obtenção de um único campo. Este tipo de ocorrências pode revelar-se difícil de reproduzir em execuções isoladas, sendo mais provável ser detetado na sequência de múltiplas execuções (e.g. em testes de carga).

Nomenclatura inconsistente com os requisitos

Havendo uma verificação integral de todos os aspetos especificados nos requisitos, torna-se inevitável a deteção de erros de ortografia, campos trocados, acentuação em falta, entre outros exemplos de inconsistências.

Defeito de Robustez

Neste item, destaca-se o ecrã de *Acessos, Digitalizações e Impressões* que possui a particularidade de exibir os dados ordenados em função do horário em que é acedido. Apesar do serviço que fornece os dados ao ecrã se encontrar protegido, para horários inválidos (fora da hora do expediente), o controlador que gera o ecrã não se encontrava totalmente protegido, sendo que nem os próprios requisitos especificavam qual deveria ser o comportamento esperado nessas situações. Ao executar os testes de fiabilidade, observou-se que entre as 0h e as 8h ocorria uma exceção genérica na abertura do ecrã, dado que a chamada ao serviço de relatórios não é válida devido ao horário em questão. Ainda que esta situação dificilmente pudesse ocorrer em contexto real, uma vez que a instituição se encontra encerrada durante esse horário (das 0h às 8h), procedemos ao seu tratamento no sentido de evitar aquela ocorrência, sendo naquelas situações mostrado o último horário válido.

Desempenho A repetição prolongada da execução da suite automática, no âmbito dos testes de fiabilidade, permitiu que fosse detetado um problema no desempenho do carregamento do ecrã - "*Detalhe do Erro* dos *Processos Atribuídos ao Administrador de Sistema*". Uma inspeção cuidada de código revelou que as chamadas ao serviço do módulo de Organização e Autorização eram as causadoras de um mau desempenho, e manifestavam um tempo de resposta inconstante entre chamadas sucessivas. Face a esta situação, a implementação do controlador do ecrã foi alterada, recorrendo-se a uma estratégia diferente de injeção de dependências, o que veio determinar a resolução do problema.

5.4.5 Limitações

Durante a especificação e implementação da suite de testes, procurou-se cobrir todos os requisitos funcionais testáveis, isto é, todos aqueles que pudessem ser verificados, direta ou indiretamente, através das ações de um utilizador. No entanto, devido a constrangimentos de ordem temporal, optou-se por não automatizar a verificação de alguns aspetos, em virtude de se ter considerado que o valor obtido não compensaria o tempo despendido na sua implementação. Assim, enumeram-se três aspetos que foram deixados de parte, mas que poderão ser implementados no futuro, permitindo alcançar uma cobertura completa:

Pesquisa com múltiplos parâmetros em simultâneo

No teste ao ecrã de *Processos atribuídos ao Administrador de Sistema* seria interessante estender o mecanismo de pesquisa para utilizar múltiplos parâmetros em simultâneo e de forma automática, resultando numa cobertura superior.

Ordenação dos dados de um ecrã segundo múltiplos critérios

Dado que apenas o ecrã de *Processos anulados e motivos de cancelamento* beneficiaria deste mecanismo de verificação, decidiu-se que o esforço para incrementar o teste genérico não se revelava como um bom investimento. Este facto, é tanto mais relevante se considerarmos que este mecanismo teria de ser suficientemente genérico para suportar diferentes tipos de dados e de comparadores recorrendo a uma configuração.

Validação de informação adicional por tipo de processo

No requisito *REQ-ECRA-PRCATRADMSIS-0060-Informação adicional por tipo de processo* (ver apêndice B), é especificada qual a informação adicional a exibir na presença de quatro tipos distintos de processos. Uma vez que este requisito lida com instâncias de processos de negócio, existe uma dificuldade acrescida em tornar o teste auto-contido, ou seja, capaz de correr em qualquer ambiente do teste do projeto. Assim, neste contexto, foi necessário identificar um conjunto de instâncias de exemplo de processos de negócio que pudessem ser carregados na *BPMS* no início do teste.

De realçar porém que, no ambiente de desenvolvimento disponível, apenas foi possível extrair dois dos quatro tipos de processo especificados neste requisito (*Entidades e Contas* e *Meios Pagamentos BFA Net*). Apesar disso, ressalta-se que o teste segue uma abordagem *data-driven*, podendo ser facilmente extensível assim que for possível obter instâncias de exemplo para os outros dois tipos de processo.

Apesar de não se ter alcançado uma cobertura total de requisitos através de testes automáticos, é importante sublinhar que uma correta gestão de *trade-offs* foi crucial para que fossem implementados outros mecanismos de maior relevância, conforme referido na secção 5.6-Valências, destacando-se o baixo acoplamento ao ambiente em teste, através de procedimentos como a abstração de utilizadores e a importação de instâncias de processos de negócio.

5.5 Testes de Robustez

Entende-se por robustez, o grau até ao qual um componente ou sistema pode funcionar corretamente na presença de entradas inválidas ou condições extremas [IEEE 610][41].

Nesta secção é descrita a motivação na conceção deste tipo de testes, estando estes organizados em duas vertentes distintas: *Interface do Utilizador* e *Camada de Serviços*.

5.5.1 Motivação

Os testes especificados na secção anterior (5.4) têm como principal objetivo testar a componente funcional do sistema, verificando se o *software* desenvolvido está de acordo com a especificação de requisitos. No entanto, este tipo de "*happy-path testing*"pouco contribui para avaliar o comportamento do sistema face a dados de entrada inválidos e a situações extremas, deixando-o sujeito a potenciais vulnerabilidades e defeitos que poderão apenas vir a ser detetados numa fase tardia do ciclo de desenvolvimento.

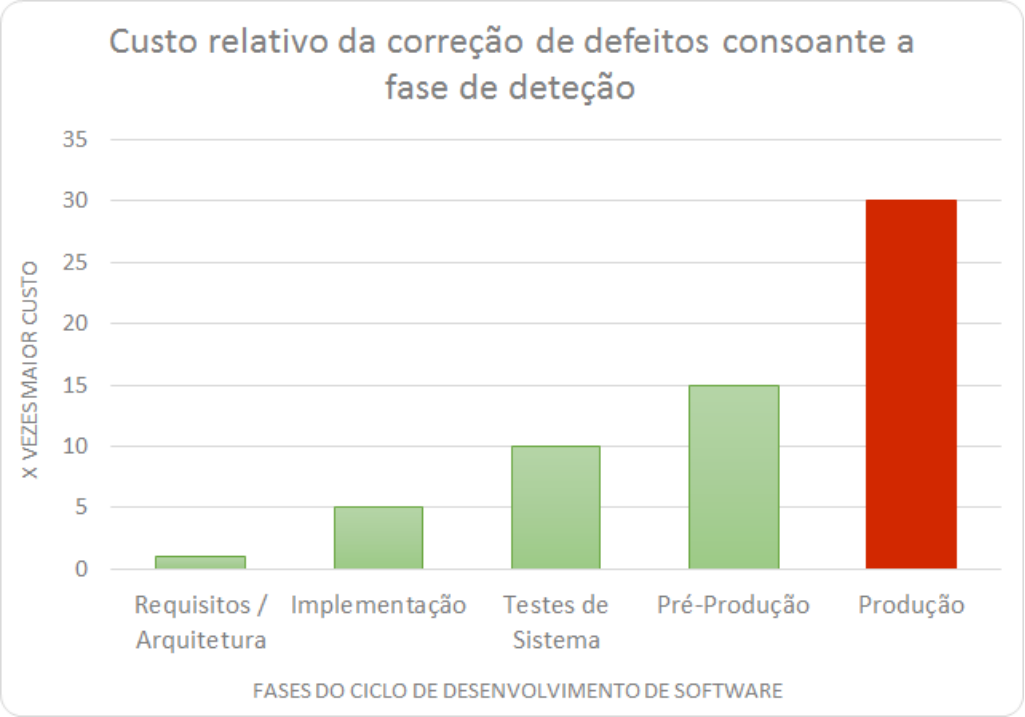


Figura 5.2: Impacto financeiro da correção de um defeito em cada etapa do *SDLC*[64] - adaptado de [NIST-2002][17]

Como se pode observar no gráfico representado na figura 5.2, quanto mais cedo se deteta um defeito, mais económica se torna a sua resolução[17]. Assim, os testes de robustez tornam-se fundamentais para melhorar a confiabilidade do sistema, favorecendo a deteção precoce de defeitos e contribuindo para a entrega de um produto de qualidade superior.

Atendendo ao contexto do projeto *eMudar*, ao âmbito do estágio e indo ao encontro do guia de Testes de Robustez para *Software Intensive Systems*[18], optou-se por incidir os testes de robustez em duas interfaces distintas: a *Interface Gráfica* e a *Camada de Serviços*. Na figura 5.3 observa-se um esquema ilustrativo da abordagem seguida pelos testes de robustez efetuados ao Módulo de Relatórios.

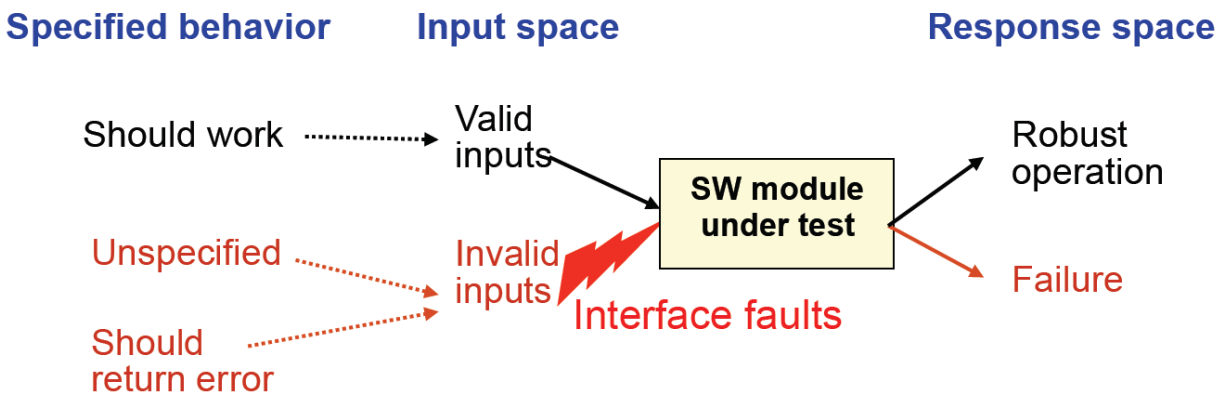


Figura 5.3: Abordagem seguida pelos Testes de Robustez efetuados[65]

Constata-se na figura 5.3, que as interfaces são exercitadas recorrendo à conjugação de parâmetros de entrada válidos e inválidos, procurando desta forma revelar defeitos nas mesmas que resultem em falhas.

Embora também pudessem ter sido efetuados testes de carga ao sistema, optou-se por não explorar esta vertente, uma vez que o ambiente de teste em utilização não está preparado para este fim e que a sua configuração de *hardware* e *software* não é equivalente à utilizada num cenário real.

5.5.2 Interface Gráfica

As interfaces gráficas são tipicamente a porta de entrada com a qual os utilizadores interagem com o sistema. De forma a fazer face aos desafios da atualidade, estas interfaces têm-se tornado cada vez mais complexas, chegando a constituir até 60% do código de um sistema[35]. Só no projeto eMudar, o *front-end* é responsável por cerca de 50% de toda a base de código do sistema, enaltecendo assim a importância desta componente. É pois imperativo que lhe seja dada uma atenção especial para garantir a sua qualidade.

No âmbito desta secção exploraram-se duas vertentes distintas:

Entradas inválidas

Neste item pretende-se avaliar a eficácia dos mecanismos de validação existentes recorrendo à técnica *Boundary Value Analysis* que se encontra exemplificada na figura 5.4.

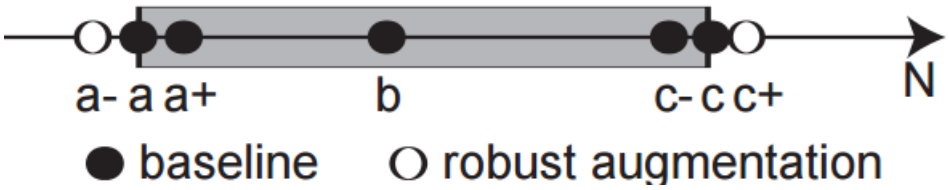


Figura 5.4: Exemplificação da divisão do espaço de entrada em casos de teste na verificação de uma dada interface[66]

Esta técnica promove a criação de casos de teste que exercitem as diversas partições que uma variável de entrada pode apresentar, focando-se no tanto nas gamas de entrada válidas, representadas a preto, como em gamas inválidas, representadas a branco[66].

Resolução do ecrã

De forma a evitar situações inusitadas, é desejável que as diferenças entre o ambiente de desenvolvimento e de produção assumam diferenças tão ténues quanto possível. Porém, diz-nos a experiência, que raramente este cenário é atingido. A resolução de ecrã utilizada não é exceção, sendo usado 1920x1080 no desenvolvimento. Porém, apesar da crescente popularidade desta resolução, ela não é a mais utilizada, sobretudo em Angola, o país em que o cliente se insere. Na figura 5.5 observamos a evolução das resoluções mais utilizadas no acesso a páginas *web* com base em estudo da StatCounter[1], um serviço de recolha de métricas e estatísticas para a *web*.

Observa-se na figura 5.5 que em 2011 (ano de início do projeto) a resolução predominante em Angola era 1280x800; no entanto, nos últimos anos a resolução 1366x768 tornou-se a mais popular, tendo a sua adoção estabilizado em 2015, apresentando uma tendência de decréscimo em 2016. A utilização de 1920x1080

[1]Página oficial do StatCount GlobalStats: `http://gs.statcounter.com`

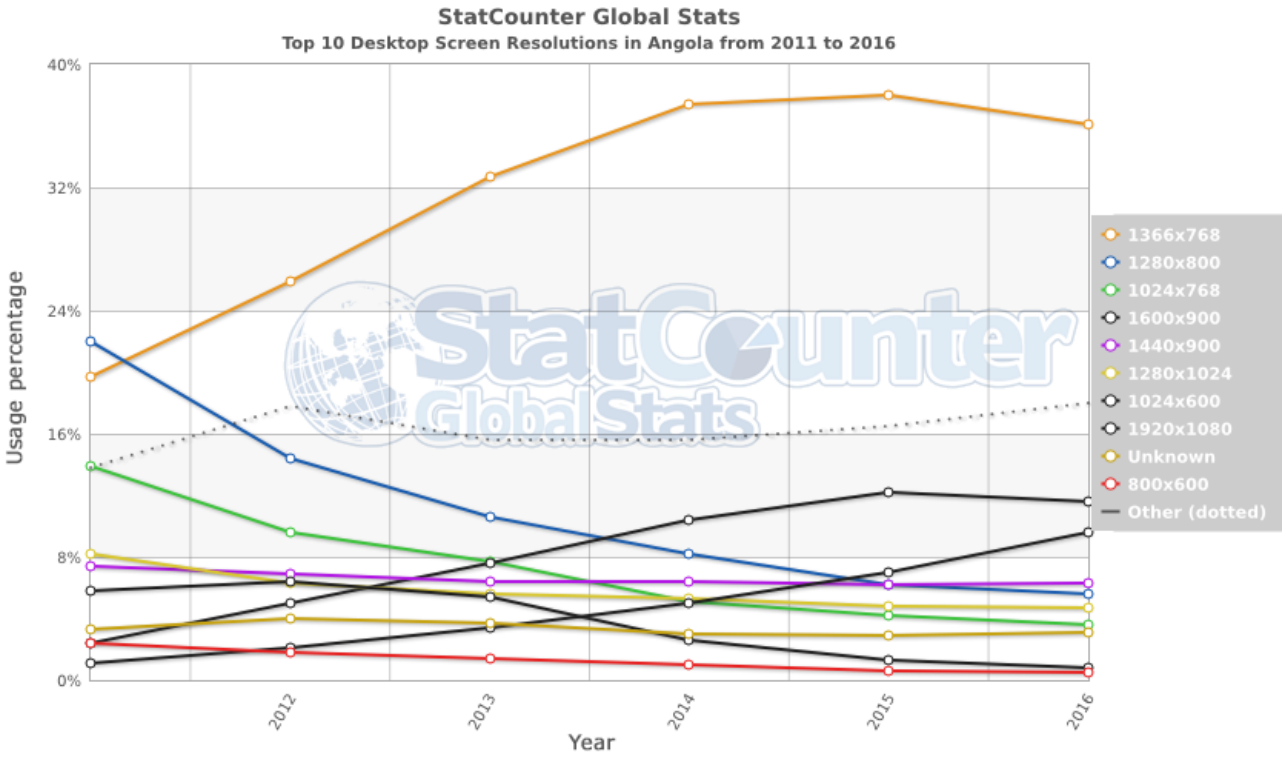


Figura 5.5: Evolução da adoção de resoluções de ecrã em Angola entre 2011 e 2016[1]

tem apresentado um crescimento gradual, sendo em 2016 a terceira mais usada. Apesar disso, mais importante do que a resolução propriamente dita é na proporção entre a largura e a altura do ecrã que se poderão encontrar as principais diferenças a nível da disposição dos elementos num ecrã.

Neste sentido, é importante que a disposição dos elementos do Módulo de Relatórios seja verificada em diversas resoluções de ecrã, com especial atenção à proporção entre a largura e altura na escolha das mesmas. Nesta tipologia de teste procura-se encontrar dois tipos de defeitos com base no seu nível de gravidade, a saber:

Defeitos de Funcionalidade - aqueles que afetam a utilidade da aplicação prevenindo o acesso a alguma função, destacando-se: botões em falta, omissão de dados, entre outros;

[1]Referência da imagem: `http://gs.statcounter.com/#desktop-resolution-AO-yearly-2011-2016`

Defeitos Estéticos - aqueles que afetam a qualidade da exibição sem comprometer o acesso a funcionalidades, tais como: desalinhamento ou sobreposição de elementos, cortes em rótulos, entre outros.

Este teste pode ser consultado na *Especificação dos Casos Teste do Módulo de Relatórios* (apêndice *confidencial* E), como o nome *STC-GUI-Robustness-MDLRL-Suporte a múltiplas resoluções*.

5.5.3 Camada de Serviços

A realização de testes de robustez à camada de serviços pode parecer, numa primeira instância, redundante, face aos testes já efetuados a nível da interface gráfica. No entanto, salienta-se que os serviços *web* desenvolvidos podem vir a ser utilizados, no futuro, por outras componentes que podem não ser robustas. Assim, atendendo a que estes serviços lidam diretamente com a camada de dados, considerou-se relevante que fossem igualmente sujeitos a este tipo de testes.

Porém, ao contrário dos testes efetuados à interface anterior, procurou-se fazer uso de uma técnica alternativa, recorrendo a uma ferramenta que permitisse automatizar o processo, contribuindo para uma robustez superior, sem consumir um elevado esforço manual para a sua aplicação[67].

Assim, fez-se uso do **wsrbench**[68], uma ferramenta desenvolvida no *DEI*, que permite automatizar a realização de testes de robustez a serviços *web* baseados em tecnologia *SOAP*. Disponibilizada sob a forma de um *SaaS*, a ferramenta recebe o endereço do descritor de um serviço (*WSDL*[1]) e faz a listagem dos seus métodos e parâmetros, conforme pode ser observado na figura 5.6.

O utilizador deve, em seguida, proceder à especificação do domínio de validade de cada parâmetro, de forma a permitir a sua validação por parte da ferramenta.

Apesar da utilização deste tipo de ferramentas não garantir a robustez absoluta de um serviço, considera-se que contribui positivamente nesse sentido, testando um conjunto interessante de cenários, com um mínimo de esforço, favorecendo a

[1]Especificação do *WSDL*: `https://www.w3.org/TR/wsdl`

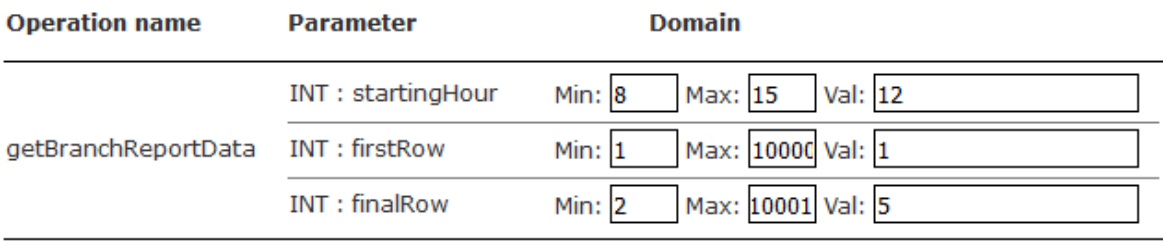


Figura 5.6: Exemplo da especificação do domínio de valores na ferramenta *wsr-bench*

deteção dos defeitos mais habituais daquele tipo. Sublinha-se, no entanto, que a sua disponibilização enquanto *PaaS*, não facilita a sua utilização no mercado empresarial, já que exige que os serviços estejam expostos para poderem ser testados, algo que nem sempre é exequível. Também se evidenciaram algumas lacunas no que diz respeito ao suporte de parâmetros compostos por múltiplos valores, como listas e dicionários, não sendo tendo sido possível especificar, nessas situações, mais do que uma unidade, impedindo assim a correta validação de alguns métodos.

5.5.4 Resultados

Nesta secção descrevem-se os principais resultados dos testes de robustez descritos anteriormente, agrupados consoante a camada em questão.

Interface Gráfica

A nível da validação dos parâmetros de entrada do ecrã de *Processos Atribuídos ao Administrador de Sistema*, testada pelo teste *STC-GUI-RBN-ECRA-PRCATRADMSIS-0030-ACÇÃO PESQUISAR* (ver apêndice E), observou-se que a implementação efetuada foi eficaz na deteção dos diversos parâmetros inválidos definidos, evidenciando ser até demasiado restritiva, em alguns casos. Face a esta situação a implementação foi ajustada, tendo posteriormente sido re-testada, tendo passado todos os testes propostos.

Relativamente ao teste de resolução, denominado *STC-GUI-RBN-MDLRL-Suporte a múltiplas resoluções* (ver apêndice E), não foram encontrados defeitos graves que

impedissem o correto funcionamento no ecrã. No entanto, foi possível observar que a ausência de suporte para a criação de interfaces responsivas, por parte da *framework ZK* utilizada no *frontend*, impede uma rentabilização do espaço disponível em resoluções superiores. Este acontecimento pode ser observado na figura 5.7. Nota: Por razões de privacidade, alguns campos foram ocultados.

Camada de Serviços

Relativamente ao serviço que fornece dados para o Módulo de Relatórios, não foi encontrado nenhum defeito de robustez, mas salienta-se que nem todos os seus métodos pudessem ser testados, devido a limitações da ferramenta utilizada já referidas. Na figura 5.8 é possível observar um resultado positivo da ferramenta para o parâmetro *startingHour* do método *getBranchReportData*.

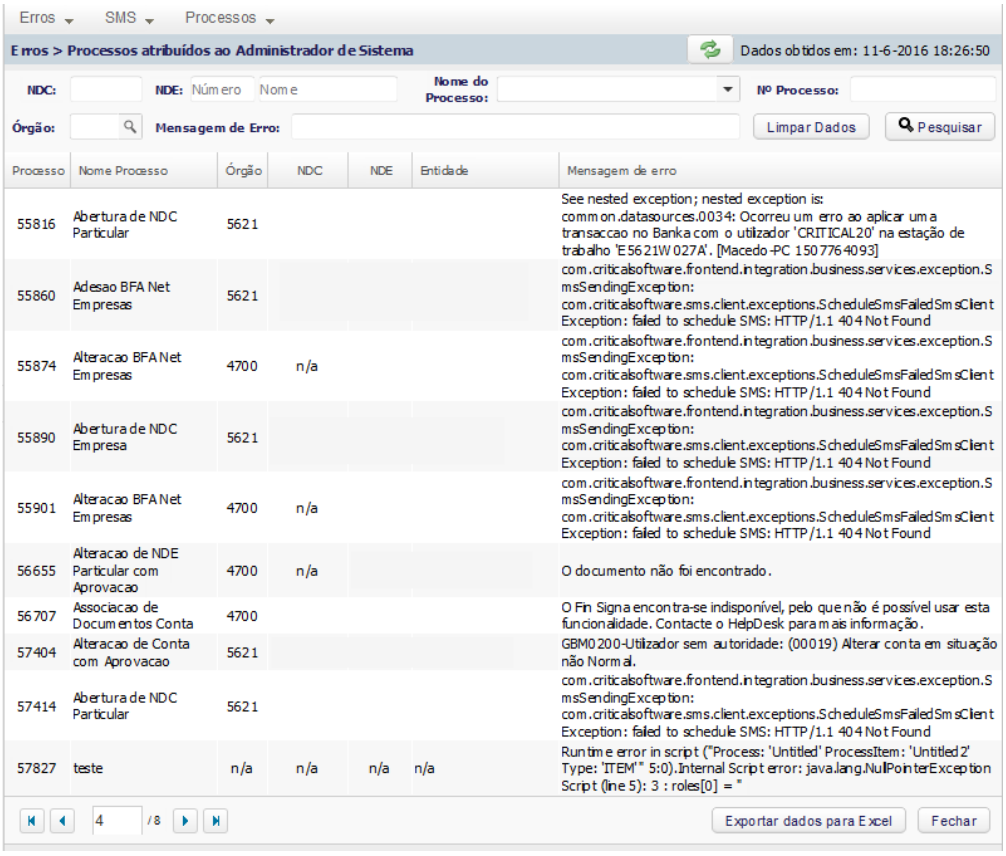
5.6 Valências

Manutenibilidade

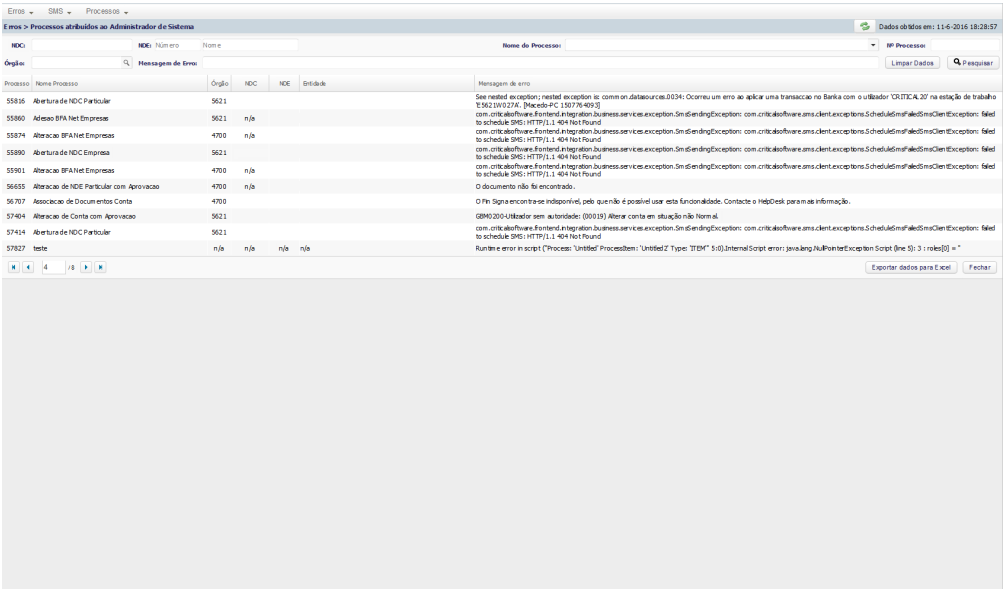
Um princípio fundamental mas que, por vezes, passa despercebido é o facto de que a Automatização de Testes é, no fundo, desenvolvimento de *software*[31][69]. Considera-se assim que, independente da estratégia ou tecnologia usada, muitos dos princípios empregues na criação de *software*, podem ser aplicados na implementação de testes automatizados[69].

Considerando que uma das principais parcelas de custo do desenvolvimento de *software* é a fase de manutenção, variando entre os 40% e os 80%[44], também a implementação de uma suite de testes automáticos está sujeita a custos de manutenção pertinentes. Sobretudo considerando o caráter cada vez mais ágil e mutável que os sistemas em teste apresentam[31].

Não sendo a automatização de testes uma "bala de prata"[70] em qualidade de *software*, é necessário fazer uso desta técnica de forma racional e consciente, procurando tirar partido das suas virtudes. Assim, é importante aceitar desde logo que os testes automáticos implicam custos de manutenção, pelo que é necessário



(a) Resolução 1024x768



(b) Resolução 1920x1080

Figura 5.7: Teste ao suporte de múltiplas resoluções do ecrã de Processos atribuídos ao Administrador de Sistema

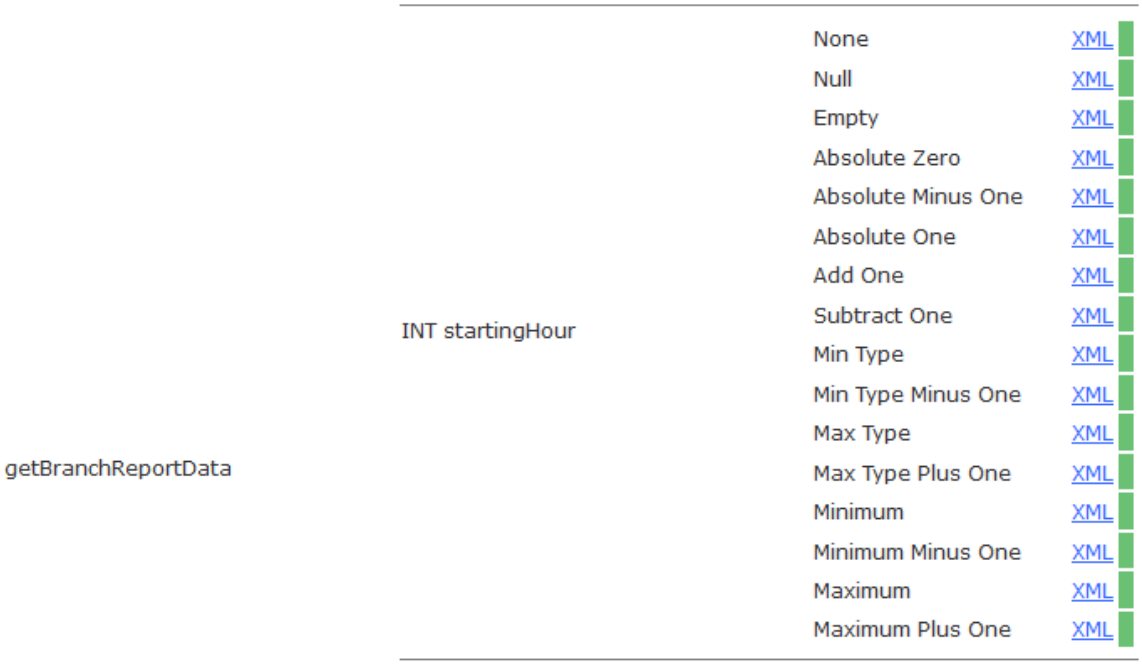


Figura 5.8: Resultado parcial de um Teste de Robustez efetuado pela ferramenta *wsrbench*

tomar diligências para manter o custo desta parcela tão reduzido quanto possível, através da aplicação das melhores práticas de engenharia e de qualidade de *software*.

Uma forma de o fazer é através da utilização de *Design Patterns*[36], conforme indicado nos dois parágrafos seguintes, e evitando a ocorrência de *code smells* no código dos testes[38][69].

Idempotência de testes

As boas práticas em testes automatizados[36] preconizam que um teste deve ser repetível quantas vezes for necessário, dispensando intervenção humana. Esta prática traduz-se no *Clean State Fixture*[38], que representa um *design pattern* de robustez de testes automatizados e cuja utilização permite concluir que um bom teste deve lidar com as suas pré-condições, de forma automática. No Módulo de Relatórios, isto é particularmente relevante no ecrã de *Detalhe de Processo*, que

lida com instâncias de processos de negócio da *BPMS* exibindo dados de acordo com o tipo de processo em causa e permitindo alterar o seu estado ou cancelá-lo.

Para verificar este ecrã, foi extraída da *BPMS* de desenvolvimento, uma instância exemplificativa para cada um dos quatro tipos de processos tratados neste ecrã (*Entidades e Contas, Cheques, Cartões, BFA Net*) que devem ser adicionados ao sistema, antes do teste. Para garantir a sua idempotência, respeitando o *pattern* referido, automatizou-se este procedimento, recorrendo a uma interface de importação de instâncias de processos de negócios, já existente, mitigando assim o risco número quatro, no âmbito da Garantia de Qualidade do Módulo de Relatórios (ver secção 5.1).

Conformidade com *Design Patterns*

De forma a superar a inexperiência do estagiário nesta área, procurou-se fazer um estudo sobre as boas práticas existentes, bem como sobre as falhas mais recorrentes, com vista à implementação de uma suite de testes eficaz e de maior qualidade.

Destacam-se assim os seguintes princípios:

- ***Single Glance Readable*** - Um teste deve apresentar uma elevada legibilidade, podendo ser facilmente compreensível por qualquer *stakeholder*[36][38]. Para ir ao encontro desta norma, procurou-se tirar partido da notação *Gherkin*[1] que, conforme referido na secção 2.3, permite descrever o comportamento de um *software* sem entrar no domínio da sua implementação.
- ***Data-Driven*** - Na presença de múltiplos testes que partilhem uma lógica semelhante, diferindo apenas a nível de dados, é recomendável que se tire partido desta técnica de reutilização para garantir uma manutenibilidade superior. Como exemplo da sua aplicação, destaca-se a implementação do teste genérico (*ATC-GUI-MDLRLTMDR-Generic*) que faz a verificação dos pontos comuns a todos os relatórios, podendo ser facilmente estendido ou instanciado recorrendo à especificação de alguns parâmetros. Também o teste *ATC-GUI-ECRA-PRCATRADMSIS-0050-Ecrã de Detalhe do Erro* faz uso deste paradigma, suportando uma validação facilitada e facilmente extensível

[1]Página oficial do *Gherkin*: `https://github.com/cucumber/cucumber/wiki/Gherkin`

```
Feature: Refund item

  Scenario: Jeff returns a faulty microwave
    Given Jeff has bought a microwave for $100
    And he has a receipt
    When he returns the microwave
    Then Jeff should be refunded $100
```

Figura 5.9: Exemplo da utilização da notação *Gherkin* na definição de um teste[1]

a múltiplos processos, conforme exemplificado na imagem 5.10 (por razões de privacidade alguns campos foram substituídos por pontos). Nela, é possível observar a referência a três instâncias de processos de negócio utilizadas para validar o ecrã de Detalhe do Erro, possuidor de um campo denominado "Informação Adicional", que apresenta dados consoante o tipo de processo.

- ***Page-Object*** - Este *pattern* tira partido das potencialidades da programação orientada a objetos, promovendo a criação de um objeto responsável pela interação com uma página ou componente do Sistema em Teste (*SUT*). Considera-se que a sua utilização facilita a criação de uma estrutura que promove uma separação clara de responsabilidades, contribuindo para a reutilização de código e a consequente manutenibilidade da suite de testes automáticos[71][72].

Apesar da adoção deste tipo de práticas contribuir para um maior custo inicial (em análise, leitura/estudo e planeamento), considera-se que a sua aplicação está associada a benefícios a médio prazo, contribuindo para um retorno do investimento na suite de testes.

Abstração de perfis, utilizadores e ambientes

Para ultrapassar a complexidade em lidar com o mapeamento variável entre utilizadores e perfis nos diversos ambientes, conforme descrito na secção 5.7.2, implementaram-se dois mecanismos de suporte.

[1]Origem da imagem: `https://cucumber.io/docs/reference`

```
PROCESS_DETAIL_DATA = {
  "Entidades e Contas": {
    'instance_file': "process_62172_02.05.2016_12.16.58.instance",
    'results': {
      'locator': u"//div[contains(@class,'z-window-modal
      z-window-modal-shadow')]//span[contains(text(), 'Informação Adicional')]",
      'value': u"NDE: ..............        NDC: ..............
      Documento de identificação: PASSAPORTE - .............."}},

  "Meios de Pagamento - Cartões": {
    'instance_file': "process_60067_02.05.2016_13.52.41.instance",
    'results': {
      'locator': u"//div[contains(@class,'z-window-modal
      z-window-modal-shadow')]//span[contains(text(), 'Informação Adicional')]",
      'value': u"Número do cartão: .............."}},

  "Meios Pagamentos - BFA Net": {
    'instance_file': "process_59504_02.05.2016_13.43.18.instance",
    'results': {
      'locator': u"//div[contains(@class,'z-window-modal
      z-window-modal-shadow')]//span[contains(text(), 'Informação Adicional')]",
      'value': u"Número de adesão BFA Net: .............."}}
}
```

Figura 5.10: Exemplo de um conjunto de dados utilizados na execução de um teste *data-driven*

Em primeiro lugar, cada teste implementado encontra-se associado ao perfil que desbloqueia o acesso à funcionalidade visada, sendo esta relação universal para todos os ambientes.

Em segundo lugar, foi implementado um procedimento que permite obter o mapeamento entre utilizadores e perfis para o ambiente em causa, armazenando-o numa representação intermédia em *XML*, promovendo uma interoperabilidade facilitada. Este procedimento, deve ser executado no arranque da suite, de forma a garantir que os testes possam fazer uso de utilizadores com o perfil necessário para as funcionalidades em teste, num determinado ambiente.

Configuração programática do *browser*

De forma a melhorar a fiabilidade dos testes de aceitação automáticos, foi necessário efetuar algumas afinações a nível do *browser*, com vista à sua adaptação para este tipo de utilização. Porém, e de forma a evitar a necessidade de alterações manuais *a priori*, procedeu-se à manipulação programática do seu perfil de execução durante o seu arranque, recorrendo-se para o efeito a um método disponibilizado pelo *WebDriver*.

Muitas das configurações utilizadas são comuns em ambientes empresariais de acordo com a *Mozilla Developer Network*[73], nas quais podemos destacar a desativação de:

- Mecanismos de auto-atualização do *browser*, extensões e *plugins*;
- Serviços de suporte desnecessários: telemetria, relatório de falhas e saúde;
- Lista negra de extensões que poderia obrigar à atualização do *Java*.

Para colmatar problemas pontuais como a exibição de mensagens de aviso do *browser*, destinadas a seres humanos, e ainda para permitir testar funcionalidades como o *download* de ficheiros, adicionaram-se também configurações com os seguintes objetivos:

- Forçar a ativação de *plugins*: e.g. Java;
- Silenciar avisos destinados a utilizadores: e.g. sair da página enquanto esta ainda se encontra em carregamento;
- Facilitar o *download* de ficheiros sem intervenção do utilizador.

Relatórios detalhados

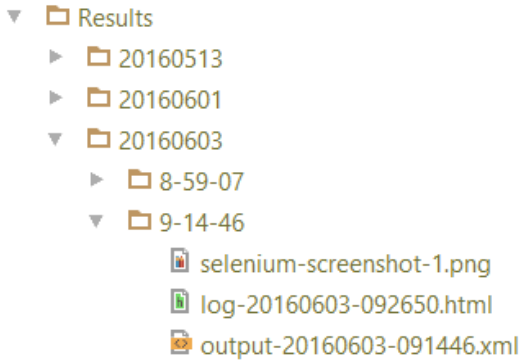
Na figura 5.11a ilustra-se a estrutura hierárquica que se definiu para o armazenamento dos relatórios gerados pela *framework*. Observa-se que para cada execução é criada uma pasta com o *timestamp* correspondente, sendo o relatório armazenado em dois formatos distintos: numa página *HTML*, facilmente legível por humanos,

e em formato *XML*[1], facilitando a sua análise por outras ferramentas externas.

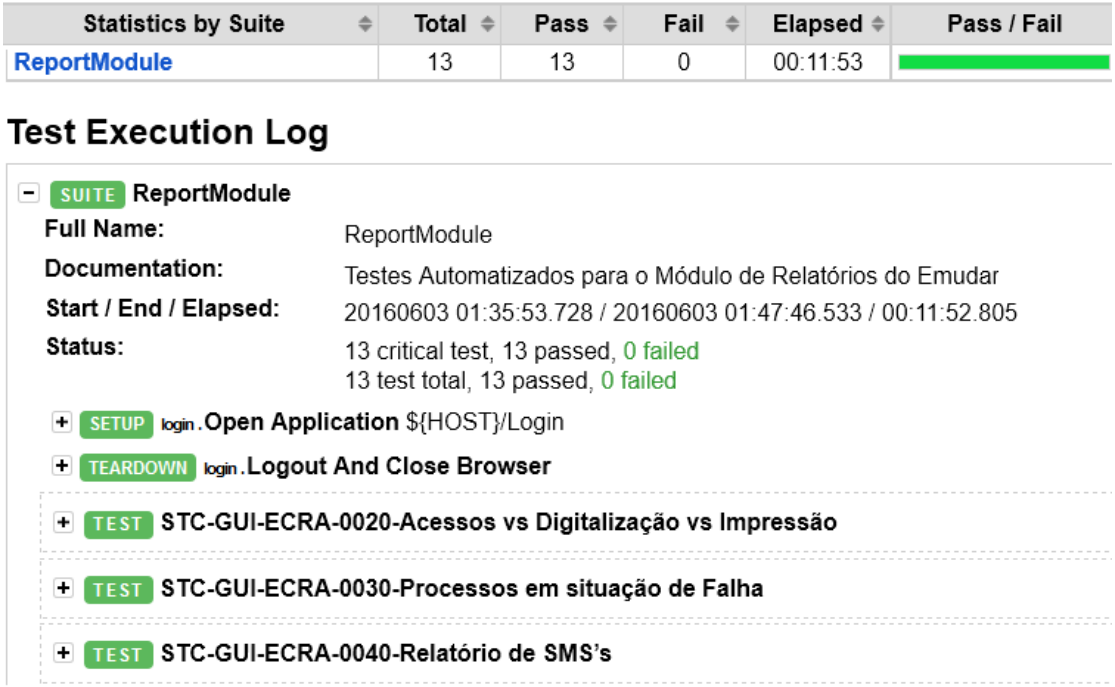
Cada uma destas execuções é guardada numa pasta correspondente à data (ano, mês, dia) em que foi executada. Salienta-se que para além do relatório são ainda armazenadas um conjunto de imagens capturadas ao longo do mesmo, facilitando a reconstrução dos passos e a rastreabilidade de um eventual defeito.

Já na figura 5.11b encontra-se representado parte de um relatório gerado, sendo visível um resumo de alto-nível da execução da suite. No entanto, é possível aceder de forma interativa, ao detalhe de cada teste, indo ao pormenor de cada *keyword*, o que facilita a identificação de um eventual defeito, sobretudo se se considerar que também estão incluídas representações periódicas do ecrã.

[1] http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#xunit



(a) Estrutura hierárquica para armazenamento dos relatórios de execução



(b) Exemplo do relatório gerado pela *Robot Framework*

Figura 5.11: Ilustração do *output* gerado pela *Robot Framework*

5.7 Desafios e Dificuldades

Nesta secção ilustram-se aquelas que foram as principais dificuldades na implementação de testes automatizados no projeto *eMudar*, agrupados segundo três categorias distintas, a saber: desafios pessoais, desafios de projeto e das tecnolo-

gias de teste.

5.7.1 Desafios Pessoais

De realçar a falta de experiência na área da Garantia de Qualidade de Software, temática que é apenas abordada, em concreto, numa unidade curricular do plano curricular de Engenharia de Software integrado no mestrado em Engenharia Informática pela UC. Não obstante, reitera-se ainda que a referida unidade curricular permitiu que fossem assimiladas as bases e fundamentos cruciais para orientar o estudo e a aprendizagem durante esta etapa do estágio.

5.7.2 Desafios do Projeto

Decorridos cinco anos da criação do projeto *eMudar*, a sua estratégia de testes nunca adotou a utilização e manutenção de uma suite de testes de aceitação automática pelo que poderá afirmar-se que o sistema não se destaca pela sua testabilidade. Com esta afirmação, não se pretende insinuar que o sistema não é testável, mas sim que, dado o seu contexto e todo o seu histórico, torna-se mais complexo e dispendioso testá-lo, quer de forma manual, quer automática.

De seguida, são enumerados um conjunto de fatores que contribuem diretamente para uma dificuldade acrescida na automatização de testes de sistema e/ou de aceitação:

Complexidade do projeto e ocorrência de falhas transitórias
Relembrando que o projeto eMudar apresenta uma *Arquitetura Orientada a Serviços* (*SOA*), é expectável que o adequado funcionamento do sistema dependa da correção das diferentes partes que o constituem. Destacam-se assim os sistemas *BANKA* e a *BPMS* que são componentes transversais partilhadas pelos diversos ambientes de teste e de desenvolvimento, tornando-se difícil alcançar condições de teste ideais, dada a ausência de isolamento total entre ambientes (ver secção 3.1.2). De forma a lidar com a instabilidade causada por falhas transitórias nestes

sistemas é fundamental a execução de testes de fiabilidade à suite de testes automáticos (descrita na secção 5.4.3), procurando identificar-se quais os locais mais propícios àquele tipo de falhas. Entre estas, permitimo-nos destacar:

- **Problemas no Login** - Falha no módulo de Organização e Autorização; Falha no acesso ao *BANKA* ou até por fecho de um balcão;
- **Perda de sessão** - Ocorre sempre que outra pessoa faz *login* com o mesmo utilizador;
- **Importação de processos** - Ocorrência de *timeouts* ao tentar adicionar novos processos à *BPMS*.

A maior parte destas situações são recuperáveis através da implementação de mecanismos que as detetem e que permitam a re-execução das operações em causa. No entanto, é importante realçar que estas situações obrigam a um acentuado dispêndio de tempo na implementação e estabilização da suite de testes automáticos. Deve sublinhar-se ainda que a utilização de mecanismos de repetição de tarefas pode também contribuir para mascarar falhas sistemáticas do sistema, resultando assim numa menor eficácia dos testes, devendo a sua utilização ser devidamente ponderada.

Divergência de utilizadores e perfis entre ambientes

Para testar uma dada funcionalidade num determinado ambiente (desenvolvimento, teste, pré-produção, etc), é necessário que exista pelo menos um utilizador com um perfil que permita o acesso à mesma. Este mapeamento entre perfis e utilizadores é da responsabilidade do *módulo de Organização e Autorização* que difere de ambiente para ambiente. Assim, na ausência de uma estrutura fixa, torna-se difícil especificar um conjunto de testes que possam ser executados em qualquer ambiente, uma vez que os utilizadores e os respetivos perfis não se encontram uniformizados.

Durante o estágio, e para ultrapassar esta limitação, foram consideradas duas abordagens distintas:

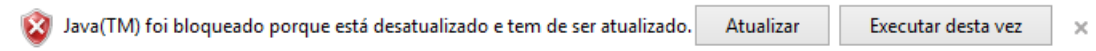
- Definição de um mapeamento pré-acordado entre utilizadores e perfis que fosse representativo do universo de funcionalidades e satisfizesse as necessidades dos *stakeholders* envolvidos no desenvolvimento, aplicando-o a todos os ambientes.
- Implementação de um mecanismo de abstração que permite associar testes a determinados perfis e não a determinados utilizadores. Ver parágrafo "Abstração de perfis, utilizadores e ambientes"na secção 5.6 - Valências.

Na nossa opinião, a primeira opção é a mais correta visto que facilita não só a criação de testes automáticos, como também a execução dos testes manuais já que o mesmo utilizador poderá ser usado independentemente do ambiente. Ainda assim, dado o seu caráter transversal e potencialmente disruptivo do fluxo de trabalho da equipa, a referida opção tem que ser implementada de forma cuidadosa e com a colaboração de todos os membros envolvidos no processo de desenvolvimento e de garantia de qualidade. Por este motivo, e devido a limitações de âmbito e de ordem temporal, optou-se pela segunda alternativa para ultrapassar este desafio.

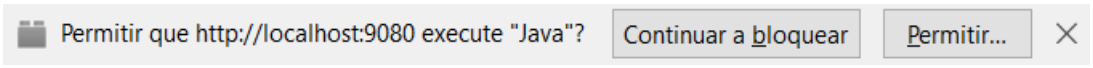
Utilização de Java *Applets*

Apesar da popularidade que esta tecnologia registou no passado, é difícil ignorar o seu forte declínio nos dias que correm[74]. As razões inerentes a tal facto devem-se às limitações que este tipo de *plugins*, assentes em *API*s legadas como o *NPAPI*, apresentam[75], podendo também estar associadas ao amadurecimento de tecnologias alternativas para a criação de aplicações *web* como o HTML5, CSS3 e JavaScript. Por tal razão, facilmente se compreende que este tipo de extensões não se devem prolongar muito no tempo[74][76][77].

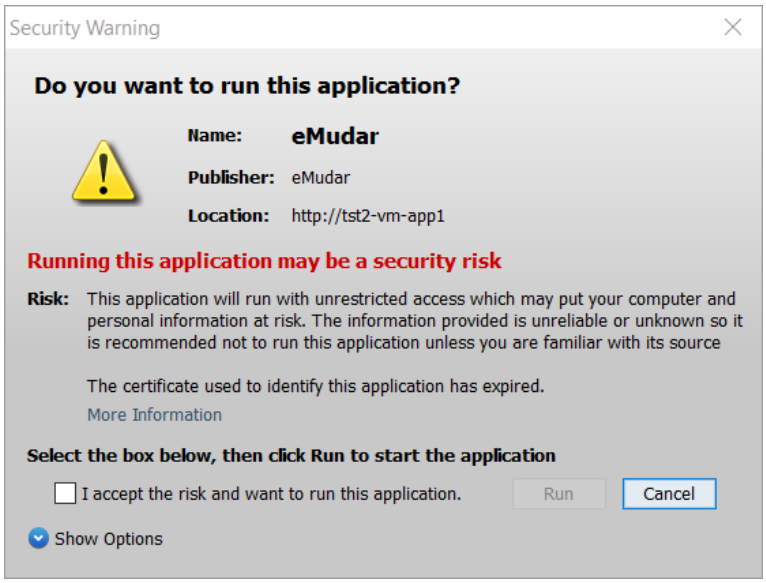
Ressalta-se que a utilização deste tipo de tecnologia em final de vida introduz complexidade acrescida no projeto, o que se traduz, entre outros, numa maior dificuldade em testar o sistema. A razão para tal facto, prende-se sobretudo com os sucessivos avisos de segurança, tanto por parte do *browser* como do próprio Java. Estes não podem ser facilmente manipulados através das *frameworks* de teste, pois não são baseados em elementos de HTML.



(a) Aviso de segurança a nível do *browser* - Internet Explorer 11



(b) Aviso de segurança a nível do *browser* - Firefox 47



(c) Aviso de segurança a nível do Java

Figura 5.12: Exemplos de avisos de segurança na execução de *Java Applets*

Na figura 5.12 podem observar-se dois tipos distintos de avisos de segurança (*browser* e *Java*). Tais avisos não são facilmente manipulados através das *frameworks* para a automação de testes, que se encontram vocacionadas para uma interação com elementos *web*.

Para mitigar esta situação procedeu-se à manipulação programática do perfil de configurações do *browser*, conforme detalhado na secção 5.6 - Valências, bem como à importação do certificado de segurança utilizado na compilação e certificação do código fonte.

Mecanismos de Segurança do ZK

A *framework* utilizada na implementação do *front-end*, o *ZK*, possui um conjunto de mecanismos de segurança contra ataques do tipo *Cross Site Request Forgery* (CSRF), uma das 10 vulnerabilidades mais críticas em aplicações *web*, de acordo com o estudo conduzido pela organização OWASP[1] em 2013[57]. Sumariamente, este tipo de ataque consiste na reutilização da uma sessão ativa numa determinada página *web*, para efetuar pedidos maliciosos, sem o conhecimento do utilizador.

De acordo com as boas práticas contra este tipo de vulnerabilidade[78][79], o *ZK* recorre a diversas técnicas[80], destacando-se a geração de identificadores aleatórios para todos os elementos *HTML* de uma página, o que dificulta a identificação dos campos necessários para falsificar um pedido.

Apesar de tornar o sistema mais seguro, esta abordagem prejudica a testabilidade do sistema ao inviabilizar a utilização do identificador de um elemento, tradicionalmente utilizado para localizar elementos *HTML*[81]. Em alternativa, foi necessário recorrer a estratégias de localização alternativas, nomeadamente a expressões *XPath* ou *CSS* de maior complexidade.

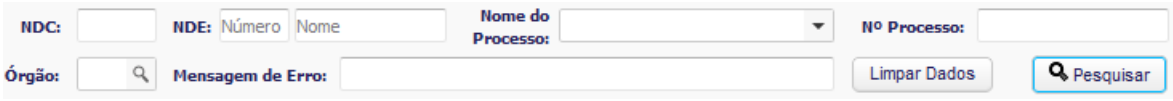
Para exemplificar esta situação, ilustra-se na figura 5.13b parte do código *HTML* gerado pelo *ZK*, onde é possível observar os identificadores aleatórios criados pela *framework*.

A título de exemplo, para aceder ao botão pesquisar é necessário recorrer a uma expressão deste género:

//div[@class='tab-workarea z-tabbox']//div[@class='z-tabpanel' and not(contains(@style, 'display: none;'))]//td[contains(text(), 'Pesquisar')]/../..

Caixa de texto 1: Exemplo de uma expressão *XPath* para aceder ao botão de pesquisa do ecrã de Processos Atribuídos ao Administrador de Sistema

[1]Página do Open Web Application Security Project (*OWASP*) - `https://www.owasp.org`



(a) Painel de Pesquisa

```
<div id="rNDPt50-body" class="z-panel-body">
 <div id="rNDPu50" class="z-panel-children z-panel-children-noheader z-panel-children-noborder">
  <div id="rNDPh60" style="display:none"></div>
  <div id="rNDPi60" class="content">
   <div id="rNDPj60" class="groupBox z-groupbox" style="background: #F9F9F9; border-color: #CFCFCF;">
    <div id="rNDPj60-cave" class="z-groupbox-cnt">
     <div id="rNDPk60" class="status" style="">
      <div id="rNDPk60-head" class="status-header" style="display: none; width: 1690px;"></div>
      <div class="status-header-bg"></div>
      <div id="rNDPk60-body" class="status-body" style="width: 1690px;">
       <table style="table-layout: fixed;" border="0" cellpadding="0" cellspacing="0" width="100%">
        <tbody style="visibility:hidden;height:0px"></tbody>
        <tbody id="rNDPt60" class="z-rows">
         <tr id="rNDPu60" class="z-row"></tr>
         <tr id="rNDPv80" class="z-row status-odd">
          <td id="rNDPw80" class="z-cell"></td>
          <td id="rNDPy80" class="z-cell"></td>
          <td id="rNDP_90" class="formLabel z-cell" colspan="3"></td>
          <td id="rNDP290" class="z-cell" align="right"></td>
          <td id="rNDP490" class="z-cell" align="right">
           <span id="rNDP590" class="z-button"></span>
```

(b) Extrato parcial do código *HTML* do painel de pesquisa

Figura 5.13: Painel de Pesquisa e fragmento de código do Ecrã de Processos atribuídos ao Administrador de Sistema

Esta expressão é composta por três parcelas distintas:

- ***//div[@class='tab-workarea z-tabbox']***
  Componente da separadores principal da interface gráfica
- ***//div[@class='z-tabpanel' and not(contains(@style, 'display: none;'))]***
  Representa todos os separadores abertos num dado momento
- ***//td[contains(text(), 'Pesquisar')]/../..***
  Representa um botão que contenha o texto "Pesquisar"

Este fenómeno torna a localização dos elementos de uma página numa tarefa complexa, morosa e, como consequência, sujeita a erros pois requer uma análise da sua

estrutura *HTML* que se torna ofuscada pela ação da *framework*.

5.7.3 Desafios das Tecnologias de Teste

A nível das tecnologias utilizadas na implementação dos testes automáticos, destacam-se as seguintes situações:

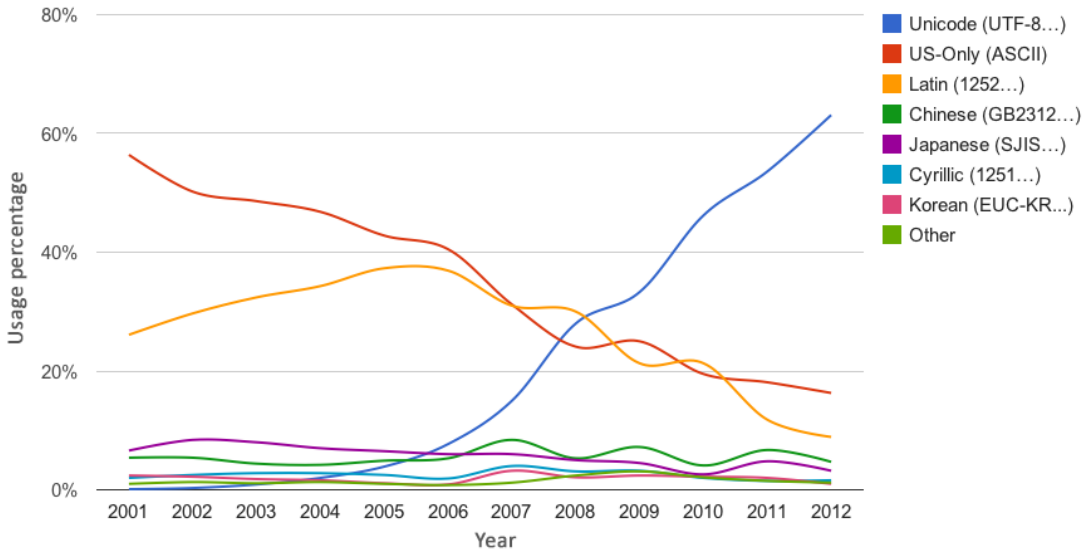
Python 2 e a codificação UTF-8
Conforme referido anteriormente, a *Robot Framework*, ferramenta utilizada para a implementação dos testes de aceitação automatizados, assenta na linguagem de programação *Python*. Devido a limitações ao nível das suas dependências, nomeadamente da biblioteca *Selenium2Library* que faz a integração com o *Selenium WebDriver* (componente responsável pela interação com os *browsers*), não suporta oficialmente, até à data, nenhuma versão da família 3.x do *Python*[1].

No sentido de resolução destes problemas de compatibilidade, foi necessário recorrer uma versão da família 2.X da linguagem. No entanto, estas versões em final de ciclo apresentam uma séria limitação que dificulta o processo de teste, pelo facto das cadeias de caracteres serem codificadas em *ASCII*, ao contrário do *Python 3* em que é utilizado *UTF-8*.

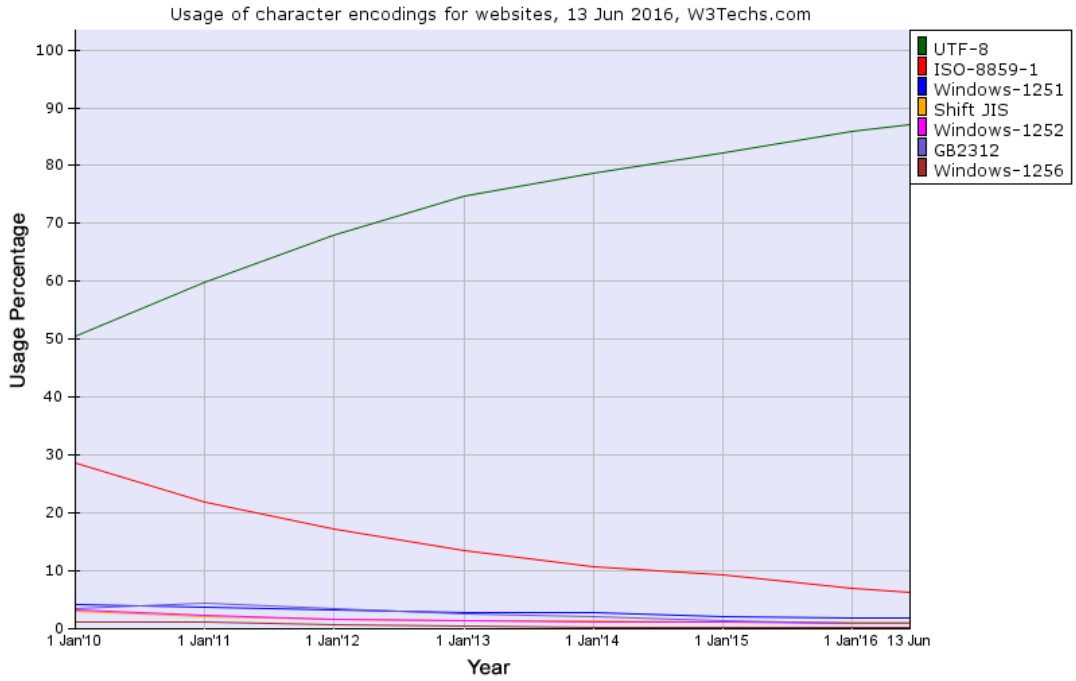
Como se pode observar nos gráficos da figura 5.14, ao longo da última década, tem-se vindo a assistir a um aumento significativo da utilização de *UTF-8* enquanto codificação predominante de caracteres em ambiente Web, alcançado cerca de 87% em Janeiro de 2016 de acordo um estudo pela *W3Techs*[2].

[1]Página onde é discutido o suporte à versão 3 do Python: `https://github.com/robotframework/Selenium2Library/issues/479`

[2]Página com o estudo da evolução de mecanismos de codificação em páginas *web*: `https://goo.gl/nebdCp`



(a) Evolução da codificação de caracteres em páginas *web* pelo *Google*[82]



(b) Evolução da codificação de caracteres em páginas *web* pela *W3Techs*[1]

Figura 5.14: Evolução de codificação de caracteres utilizada em páginas *Web* ao longo dos anos

[1]Referência para a imagem: https://w3techs.com/diagram/history_overview/character_encoding/ms/y

Assim, é seguro assumir que a utilização de *Python* 2, para lidar com páginas *web* com codificação de caracteres *UTF-8*, se pode revelar problemático[83]:

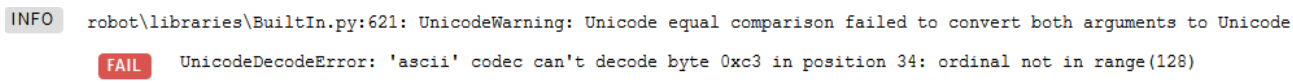


Figura 5.15: Exemplo de erros devido a problemas com a codificação de caracteres

Este tipo de ocorrências leva a que seja necessário implementar alguma lógica adicional para complementar algumas das limitações identificadas em funções primitivas da *framework*, como a comparação de cadeias de caracteres. Uma vez que duas cadeias com o mesmo conteúdo mas com codificações diferentes são, internamente distintas, resultam na falha injustificada de alguns testes.

Comportamentos irregulares da *framework* de testes

Por vezes, a falha de um teste pode ser atribuída a um defeito nas tecnologias utilizadas no processo de teste e não ao sistema em análise. Este tipo de situação é altamente inconveniente e afeta negativamente a confiabilidade na suite de testes.

Durante a fase de execução dos testes de fiabilidade à suite (secção 5.4.3), destacou-se uma falha ocasional e silenciosa no clique num elemento *HTML*. Esta situação ocorreu durante a operação de pesquisa do ecrã de *Processos atribuídos ao Administrador de Sistema* onde, por vezes, o clique no botão pesquisar não é efetuado, sem que isso dê origem a algum erro ou aviso. Perante esta situação, o teste falha pois o sistema não exibe os resultados de pesquisa como seria de esperar.

Face a este imprevisto, foram feitos esforços para determinar a origem do problema, tendo sido observado o seguinte:

- Nesta situação não eram feitos pedidos ao *web service* que fornece os dados ao ecrã;
- A inspeção de código à implementação do evento despoletado pelo botão, não revelou qualquer defeito;
- Os testes manuais à funcionalidade não permitiram reproduzir o problema;

- É possível encontrar na Internet diversas referências (Github[1], Stackoverflow[2], Stackexchange[3]) a este tipo de defeito (falha no clique de um elemento *HTML*) em relação às *frameworks* de teste utilizadas. Salienta-se que sendo um problema difícil de reproduzir, e de comportamento variável entre *browsers*, torna-se difícil diagnosticar a sua origem.

Ainda que todos os dados apontem no sentido de que o defeito seja originado pelas *frameworks* utilizadas nos testes automáticos, em rigor, não é possível afirmá-lo de uma forma peremptória.

Para colmatar este tipo de situações, procurou-se modificar a forma como se interage com um dado elemento *HTML*; no entanto, a única opção que se mostrou eficaz foi a introdução de um mecanismo de tentativas, sendo o clique repetido (no máximo uma vez), caso não seja apresentado o efeito esperado. Apesar de solucionar o problema, este tipo de práticas para ultrapassar alegadas limitações da *framework*, podem contribuir potencialmente para a redução da eficácia na deteção de defeitos no sistema, pelo que devem ser utilizadas com prudência.

5.8 Trabalho Futuro

Nesta secção, descrevem-se um conjunto de tópicos de elevado interesse na ótica da Garantia de Qualidade que não puderam, porém, ser desenvolvidos durante o estágio. Por uma questão de legibilidade, foram agrupados de acordo com o o seu âmbito nas seguintes sub-secções:

5.8.1 Automatização de Testes

Registo automático de execuções na plataforma *JIRA*
Conforme referido no capítulo 3, prevê-se que o resultado da execução de um teste seja introduzido no *JIRA*, ficando assim acessível aos membros da equipa.

[1] https://github.com/seleniumhq/selenium/issues/1202
[2] http://stackoverflow.com/questions/31725033/selenium-click-not-always-working
[3] http://sqa.stackexchange.com/questions/2333/selenium-2-element-click-is-unreliable

Relembra-se que esta prática visa assegurar a rastreabilidade entre testes, requisitos e eventuais defeitos, sendo igualmente relevante para testes manuais como para testes automáticos. Assim, propõe-se o desenvolvimento de um *plugin* que tire partido das capacidades de *reporting* da suite de testes, permitindo o seu registo automático na plataforma, dispensando a atividade humana no ciclo de vida da execução de testes automatizados. Sugere-se ainda que este *plugin* seja implementado na linguagem de programação *Python*, garantindo a interoperabilidade com a *Robot Framework* (ferramenta em utilização para os testes de sistema), tirando partido da *API* de *REST* do *JIRA*, de forma a registar programaticamente os resultados da execução dos testes.

Complemento dos mecanismos de Integração Contínua

A integração contínua visa garantir que qualquer alteração no código fonte seja rapidamente integrada e validada de forma transparente e automatizada. Na presença de uma alteração que resulte numa falha, o processo de integração contínua deve fornecer um *feedback* de forma imediata e precisa, facilitando a sua correção[84]. O *Jenkins* é uma ferramenta popular de integração contínua implementada em *Java* e disponibilizada em regime de código aberto. No contexto do projeto, encontra-se integrado com o sistema de controlo de versões, garantindo assim que o código fonte compila com sucesso à medida que vai sendo modificado. Sugere-se que se instale o *plugin* de integração com a *Robot Framework*[1] de forma a agendar a execução automática dos testes de sistema. Atendendo a que a execução de testes é um processo potencialmente demorado, considera-se que não fará sentido fazê-lo a cada alteração de código, pelo que se recomenda a sua execução por uma rotina fora da horário de expediente (madrugada e fim de semana).

Análise de Técnicas mais robustas de identificação de elementos *web*

Durante o levantamento do estado da arte relativo a ferramentas para a automatização de testes, descrito na secção 2.3, observou-se que algumas das soluções comerciais existentes recorriam a mecanismos de lógica difusa, na identificação de

[1]Página oficial do *plugin* de *Jenkins* para integrar com a *Robot Framework*: `https://wiki.jenkins-ci.org/display/JENKINS/Robot+Framework+Plugin`

elementos *web*. Considera-se que este tipo de técnica pode contribuir para uma maior resiliência dos testes automáticos, face a alterações no sistema em teste, garantindo assim robustez e manutenibilidade superior.

Paralelização dos Testes de Aceitação Automáticos

Um dos problemas colocados à adoção crescente da utilização de testes automáticos tem a ver com a morosidade da sua execução. Tal facto, contribuirá para que sejam utilizados com menos frequência, limitando portanto a sua eficácia. No sentido de inverter esta situação, é fundamental que se estudem e criem mecanismos para paralelizar a execução de testes, com intuito de reduzir o tempo necessário à sua execução. Ainda que estes testes possam ser agrupados em suites independentes, que permitam uma execução simultânea, subsiste no entanto um sério problema de conflito de utilizadores, uma vez que cada um deles só pode ter uma sessão aberta em simultâneo. Assim, observa-se que caso o mesmo utilizador fosse usado por diferentes testes em simultâneo, ocorreriam perdas sucessivas de sessão originando a falha em pelo menos um deles. Procurando evitar esta situação, sugere-se a implementação de um mecanismo transversal de gestão e alocação de utilizadores. A presente suite já faz alguns preparativos nesse sentido, associando os testes a perfis e não aos utilizadores. Recomenda-se que esta abordagem seja complementada com um mecanismo de *lock*, que reserve um utilizador disponível, com um determinado perfil, evitando que outros testes o possam utilizar, sendo libertado no final do teste.

Etiquetação de Testes

A *Robot Framework* permite que se atribuam etiquetas (*tags*) a testes, contribuindo para a distinção de diferentes tipos de teste. Esta funcionalidade permite, por exemplo, distinguir entre testes rápidos (*smoke tests*) de outros mais complexos, concedendo a oportunidade de escolha face às diferentes necessidades. De realçar que esta prática poderá ser muito útil na interação com um sistema de integração contínua, ao permitir que se distingam entre testes rápidos e/ou outros mais completos, de forma a agendar diferentes períodos de execução com base nas necessidades do projeto.

Documentação de *keywords* reutilizáveis

Para assegurar a viabilidade e manutenibilidade de uma suite de testes automáticos é necessário promover a reutilização das *keywords* que constituem os testes, contribuindo para evitar a duplicação de código[36]. No entanto, ainda que estas componentes sejam desenvolvidas nesse sentido, é fundamental que se produza documentação de qualidade e com impacto, acessível a todos os elementos da equipa. De destacar que a *Robot Framework* possui mecanismos que facilitam a produção dessa mesma documentação. No entanto, é necessário que esteja previsto um intervalo temporal para a redação e atualização da mesma, durante o desenvolvimento dos testes. De salientar também que esta tarefa de produção documental, em formato *web*, se poderá associar ao fluxo de integração contínua do projeto, garantindo que a mesma se mantenha alinhada com o conteúdo dos testes, dispensando a necessidade de intervenção humana adicional na sua atualização.

Resolução das limitações identificadas

De forma a alcançar uma cobertura ainda maior dos requisitos, sugere-se a extensão da suite de forma a colmatar limitações identificadas na secção 5.4.5-Limitações.

5.8.2 Testes de Robustez

De forma a rentabilizar o tempo disponível para a concretização deste tipo de testes, foi necessário definir um foco, tendo-se optado por incidir sobre a *Interface Gráfica* e a *Camada de Serviços*, conforme descrito na secção 5.5.1. No entanto, salienta-se que existem ainda múltiplos aspetos que poderiam ser alvo deste tipo de testes, nomeadamente a nível da camada de dados e camada de rede[18].

Assim, considera-se que seria interessante recorrer a técnicas como a injeção de falhas de forma a avaliar o comportamento do sistema, face às seguintes situações:

- Perda de ligação à base de dados;
- Corrupção / inexistência de registos necessários da base de dados;
- Corrupção / omissão de configurações.

5.8.3 Outros

Análise Estática de Código

Entende-se por análise estática a avaliação de artefactos, produzidos durante o desenvolvimento de *software*, sem que se proceda à execução dos mesmos[41]. Ao ser aplicada ao código fonte, recorrendo geralmente a uma ferramenta de suporte, permite:

- Análise de conformidade com convenções e normas de codificação, tanto gerais como específicas a um projeto / organização;
- Recolha de métricas de qualidade como a complexidade ciclomática[41], percentagem de código coberto por testes, percentagem de código duplicado, entre outros;
- Deteção de potenciais defeitos a nível do código, tais como: fugas de memória, vulnerabilidades, código morto[1], entre outros.

Considera-se que, as informações disponibilizadas por este tipo de ferramentas contribuem para a melhoria da qualidade do código e, por consequência, também do produto, podendo ser facilmente adotadas sem necessidade de qualquer investimento, dada a existência de diversas ferramentas gratuitas[85].

5.9 Conclusões

Com a massificação no acesso às tecnologias de informação, os utilizadores têm vindo a tornar-se cada vez mais exigentes, contribuindo para o estabelecimento de um mercado cada vez mais competitivo. Não obstante, a Garantia de Qualidade revela-se uma fase cada vez mais preponderante do *SDLC* de forma a satisfazer as expectativas crescentes que o *software* enfrenta.

Ao longo desta etapa do estágio, foi possível contactar de perto com as inúmeras vantagens e potencialidades da área, estimulando o desenvolvimento de um sentido crítico. Constatou-se, assim, o desafio de rentabilizar o tempo disponível para

[1] Código que não se pode atingir e, por isso, impossível de ser executado[41]

maximizar o valor obtido, sobretudo considerando os inúmeros tipos de teste e de técnicas de verificação de *software* existentes.

Automatização de Testes

O desenvolvimento da suite de testes automáticos para o *Módulo de Relatórios* permitiu evidenciar diversas lacunas do sistema *eMudar* que dificultam a sua testabilidade. No entanto, considera-se que a prova de conceito elaborada vem viabilizar a abordagem seguida, tendo conseguido mitigar os principais desafios encontrados.

Estimativas

Assim, e tal como no capítulo anterior, ocorreram algumas derrapagens a nível do planeamento, devido ao surgimento de tarefas que não foram previstas inicialmente, tais como:

- **Fase de estabilização / finalização dos testes** - Descrita na secção 5.4.3, vem favorecer a integração futura da suite com os mecanismos de Integração Contínua, em utilização no projeto;
- ***Refactoring* do código dos testes** - Considera-se que a realização desta tarefa, numa fase final do desenvolvimento dos testes, seria benéfica, tirando partido de um conhecimento superior, tanto do problema, como das tecnologias em causa;
- **Documentação dos testes automáticos para a empresa** - De forma a garantir a manutenibilidade da suite é necessário que seja elaborada alguma documentação útil para o efeito.

Finalmente, sublinha-se que o processo de Garantia de Qualidade seguido, permitiu que fossem encontrados e corrigidos alguns defeitos, considerando-se que foi útil para promover a disponibilização de uma componente de *software* mais confiável e robusta.

Capítulo 6

Análise Funcional - Módulo de Organização e Autorização

Neste capítulo é descrita a participação na análise funcional do módulo de Organização e Autorização. Este módulo, que já se encontra em produção, foi alvo de múltiplos pedidos de alteração por parte do BFA, sendo necessário proceder à atualização da especificação de requisitos existentes, tendo ficado a componente de implementação fora do âmbito deste estágio.

A concretização desta tarefa passou pela elaboração de protótipos de interface de forma a ilustrar as funcionalidades visadas, sendo posteriormente utilizados na criação de um *storyboard*, concretizado no apêndice F. Este, é composto por protótipos de média fidelidade, permitindo simular o comportamento do sistema, sendo particularmente útil para efeitos de validação, tanto por parte da equipa, como do cliente. Posteriormente, foi feita a atualização do documento de especificação de requisitos, o que envolveu a criação, modificação e remoção de requisitos.

Atendendo à elevada extensão da especificação deste módulo, que supera uma centena de páginas, é necessário compilar um resumo das alterações efetuadas de forma a contextualizar a equipa de desenvolvimento. Salienta-se que o sucesso desta tarefa passa não só pela sua eficácia, através da salvaguarda das necessidades do cliente, mas também pela sua eficiência, isto é, garantir uma análise

completa e minuciosa de forma a simplificar o trabalho da equipa de desenvolvimento, contribuindo para uma análise de impacto mais realista e minimizando eventuais dúvidas e problemas no futuro.

6.1 Processos de Negócio Visados

Antes de proceder à análise dos pedidos de alteração, é necessário adquirir algum contexto acerca do funcionamento do módulo, favorecendo uma melhor compreensão das necessidades do cliente e das limitações do módulo. Assim, é feita nesta secção uma breve descrição sobre os três processos de negócio visados nesta componente do estágio.

Manutenção de Utilizadores
Este processo é responsável pela gestão dos utilizadores do sistema *eMudar*, permitindo fazer a sua criação, pesquisa, listagem e manipulação de atributos.

Inibição de Colaborador
Este processo tem como objetivo habilitar ou desabilitar o acesso ao sistema de um determinado colaborador, seja por razões definitivas (despedimento) ou temporárias (férias, suspensão, etc). Ao nível de pesquisa aceita apenas o número de um utilizador, apresentando os seus principiais atributos e respetivas funções nos órgãos do banco.

Manutenção de Organização
Neste processo é possível gerir a hierarquia de órgãos do banco, permitindo importar novos membros do sistema *BANKA* e proceder à manipulação dos seus atributos.

6.2 Resumo das necessidades identificadas

Nesta secção é feita uma síntese dos pedidos de alteração do BFA assim como a respetiva análise das necessidades subjacentes.

Normalização da Pesquisa de Colaboradores
Durante a análise ao módulo, observou-se que existiam múltiplas variações da interface de *Pesquisa de Colaboradores* em processos como: Manutenção de Utilizadores, Inibição de Colaboradores, Transferência de Órgão e Transferência de Funções. Estas, são semelhantes entre si, apresentando diferentes tipos de filtros e listagens de colaboradores com diferentes níveis de detalhe. Conclui-se que a falha na identificação do carácter transversal da funcionalidade de *Pesquisa de Colaboradores*, resultou na ausência da sua uniformização em todo o Módulo de Organização e Autorização, afetando negativamente a sua manutenibilidade e testabilidade. Para alem disso, também a usabilidade é afetada, favorecendo a criação de *CRs* no futuro para colmatar as limitações sentidas pelo BFA. De forma a endereçar os problemas observados e a satisfazer as necessidades do banco, dever-se-á uniformizar esta funcionalidade, disponibilizando uma interface unificada que permita tanto uma pesquisa rica (através dos múltiplos parâmetros que constituem um utilizador), como um ecrã de detalhe de utilizador que garanta a visualização dos seus atributos.

Reformulação da Criação de Utilizadores
Relembrando que o sistema *eMudar* se encontra construído sobre o sistema legado *BANKA*, destaca-se que a criação de uma conta para um colaborador é um processo com duas etapas, devendo o mesmo ser criado no sistema legado, e só depois no *eMudar*, sendo a associação feita através do número do utilizador.

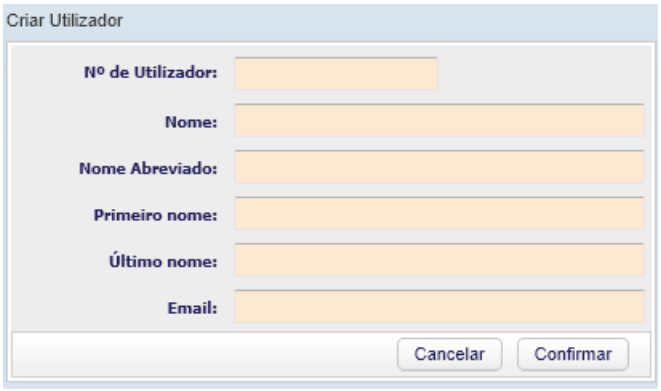


Figura 6.1: Processo de Manutenção de Utilizadores - Interface atual de criação de utilizador

Conforme pode ser parcialmente observado na figura 6.1, a atual interface de criação apresenta alguns problemas:

- Não interage com o *BANKA* no sentido de validar a existência do utilizador, ficando sujeito a enganos;
- Não previne a divergência dos dados de um utilizador entre os dois sistemas, requerendo a introdução manual de campos previamente inseridos no *BANKA*;
- Ausência de dados relevantes, como por exemplo o órgão inicial do colaborador;
- Requer a introdução de quatro campos para o nome de um utilizador.

Complemento da Situação de um Colaborador

Atualmente, um colaborador da instituição pode apresentar apenas um de dois estados perante o sistema: ativo ou inibido. Realça-se no entanto, que este esquema não permite distinguir restrições de acesso temporárias, como férias ou uma suspensão, de restrições permanentes como um despedimento de um funcionário. Assim, é solicitado pelo BFA que o *eMudar* permita fazer a distinção entre estes dois cenários, recorrendo às designações inibido e inativo, respetivamente.

Melhorias na Gestão de Órgãos
No âmbito do processo de Manutenção da Organização, foram solicitados as seguintes alterações com vista a um aumento da usabilidade e da flexibilidade do processo:

- Estender a pesquisa de órgãos para considerar todo o universo e não apenas a hierarquia selecionada;
- Admitir a definição do órgão hierarquicamente superior durante a importação e a modificação de um órgão;
- Permitir a inativação de um órgão.

Mudanças de Terminologia
No desenvolvimento de um sistema de informação é fundamental que seja utilizada a terminologia mais natural, tanto para o cliente como para o negócio. Neste sentido, o BFA solicita que seja alterado o nome de alguns processos e campos para designações mais adequadas.

6.3 Análise Funcional

A concretização desta tarefa pode ser consultada no apêndice G; no entanto, relembra-se que o referido documento não é da autoria integral do estagiário, sendo-lhe atribuída apenas a atualização e a modificação dos requisitos relacionados com os processos de negócio referidos na secção 6.1, incluindo algumas funcionalidades transversais como a Pesquisa de Colaboradores. Relembra-se que a análise funcional formalizada neste documento tem por base o *storyboard* que foi realizado pelo estagiário, podendo ser consultado no apêndice F.

6.3.1 Racional e Preocupações

No seguimento da análise efetuada ao funcionamento do módulo de Organização e Autorização, à sua especificação de requisitos e aos pedidos de alteração efetuados

pelo BFA, identificaram-se como prioritários a execução dos seguintes pontos:

- Contribuir para a atualização e manutenção da especificação de requisitos, promovendo uma maior uniformização de funcionalidades dentro do módulo, resultando uma maior reutilização de código e consequentemente maior testabilidade[36];
- Dotar o cliente de uma maior autonomia na gestão da sua organização, contribuindo para a redução de intervenções de suporte;
- Assegurar uma maior consistência de dados entre os sistemas *eMudar* e *BANKA*, a nível de utilizadores, reduzindo e limitando a intervenção humana na sua sincronização;
- Garantir um maior grau de completude dos requisitos, através da explicitação dos perfis que permitem acesso a uma dada funcionalidade, e indicando as verificações que devem ser feitas a dados de entrada.

Pedidos de Esclarecimento ao BFA

Em Engenharia de Requisitos é importante que se promova, tanto quanto possível, uma separação clara entre o domínio do problema e o domínio da solução[86]. No entanto, atendendo às restrições relativas à localização do cliente, identificadas na secção 1.5, destaca-se que o impacto na qualidade e eficácia da comunicação com o mesmo, não favorece o cumprimento deste princípio. Assim, de forma a compensar a ausência de um contacto direto com o cliente, foram efetuados dois pedidos de esclarecimento (ver o processo de *Clarification Request* descrito na secção 3.1.3) com o objetivo de obter um melhor entendimento acerca das intenções que motivaram o *CR* efetuado.

6.3.2 Resumo das alterações a serem feitas

De forma a facilitar a futura execução de tarefas, como a análise técnica (ver secção 3.2.5), estimação e consequente implementação, optou-se por efetuar um

resumo das alterações a serem efetuadas. Este, foi dividido segundo quatro categorias, uma para cada um dos processos referidos na secção 6.1, e uma categoria adicional para as alterações de carácter transversal. Esta informação foi introduzida na plataforma *JIRA* sob a forma de descrição das tarefas de implementação, promovendo uma elevada acessibilidade por parte dos membros da equipa que vierem a ser destacados para a sua concretização. Estes resumos incluem a seguinte informação:

- Descrição das alterações a serem elaboradas sob um ponto de vista técnico, incluindo imagens de ecrã que auxiliam a sua compreensão, ilustrando tanto o comportamento atual como o novo comportamento esperado (*mockups*);
- Fornecimento de dados úteis para a análise de impacto e de estimativas, destacando alguns pontos que poderão ser afetados, tais como: esquema de dados, *web-services*, processos de negócio na *BPMS*, entre outros;
- Menção dos ecrãs que devem ser descontinuados em detrimento de novas interfaces transversais;
- Referência aos requisitos afetados.

Considera-se que a realização desta tarefa promove a transmissão do contexto adquirido durante a participação na análise funcional deste módulo, contribuindo positivamente para uma maior eficácia das fases posteriores do *SDLC*.

6.4 Desafios e Dificuldades

Documentação desatualizada

Apesar do carácter transversal do módulo, muitas das funcionalidades que atualmente se encontram disponibilizadas foram criadas com o objetivo de auxiliar os membros da equipa de suporte do *eMudar* na realização de tarefas rotineiras. Com o passar do tempo, passou a ser utilizado diretamente pelo BFA, sem que o seu desenvolvimento tivesse sido diretamente faturado àquela instituição bancária. Desde então já foi alvo de vários *CRs*; no entanto, os seus requisitos e testes deixaram de ser atualizados a partir do final do ano de 2014. Esta situação dificulta

a escrita de requisitos e testes no âmbito de novos pedidos de alteração, dado que não existe uma base de trabalho atualizada. Assim, foi necessário começar por atualizar a especificação relativa aos três processos visados no estágio, antes de poder dar resposta às alterações solicitadas pelo cliente.

Interação com o Cliente

Conforme referido nas condicionantes do estágio (secção 1.5), a distância ao cliente é um fator que afeta negativamente a qualidade e rapidez da comunicação com o mesmo, constituindo um obstáculo ao processo de levantamento de requisitos. Assim, e face ao surgimento de dúvidas destinadas ao BFA, o processo da empresa prevê a criação de um pedido de esclarecimentos (ver o processo de *Clarification Request* descrito na secção 3.1.3) com vista à obtenção de informação adicional. Salienta-se, no entanto, que apesar da sua utilidade, este tipo de comunicação é bastante limitado, assentando apenas em comunicação não verbal cujo tempo de resposta pode oscilar entre dias ou até semanas.

6.5 Trabalho Futuro

Conforme referido na secção 6.4, a especificação de requisitos do presente módulo é longa (cerca de 116 páginas) e encontra-se desatualizada face à sua atual implementação. Assim, de forma a erradicar este estado de indefinição de requisitos, optou-se por dividir os diversos processos de negócio que constituem o módulo, por um conjunto reduzido de membros da equipa, garantindo um progresso de revisão e atualização mais agilizado. No entanto, salienta-se a necessidade de se proceder, *a posteriori*, a uma compilação manual dos requisitos dos diversos processos numa especificação única. Esta tarefa é fundamental para concluir a reposição da especificação de requisitos deste módulo, originando um novo documento que servirá como base para suportar novos pedidos de alteração no futuro.

Posteriormente, será necessária a aprovação da especificação de requisitos atualizada, por parte do BFA, de forma a que se possa avançar para as fases de atualização da especificação de casos testes e implementação.

6.6 Conclusões

A participação neste módulo contribuiu para a obtenção de uma consciencialização superior acerca da importância e do impacto que os requisitos têm no desenvolvimento de um sistema. Assim, identificam-se ainda três pontos a ter em consideração:

Implicações dos Requisitos
Observou-se que as decisões tomadas a nível dos requisitos afetam significativamente a construção do sistema. Assim, considera-se que certo tipo de atributos de qualidade como a manutenibilidade e a testabilidade devem ser salvaguardados a nível da especificação de requisitos[31], promovendo práticas como a reutilização de código.

Atualização da documentação
A nível processual foi evidente a importância das tarefas de atualização da especificação de requisitos e do caderno de testes no ciclo de vida de qualquer *CR*[87]. Ainda que o adiamento destas etapas possa constituir uma pequena poupança a curto prazo, considera-se que resultará num custo superior a médio/longo prazo, para manter a documentação atualizada, de forma a permitir uma correta estimação de alterações futuras.

Rastreabilidade
Também a nível de rastreabilidade, para além do mapeamento entre requisitos e testes, deve também ser feito um mapeamento entre *CRs* e os requisitos afetados, de forma a garantir que cada alteração possa ser associada a um dado pedido e ao respetivo autor. Este tipo de prática é sobretudo útil para a resolução de problemas quando existem, por parte do cliente, visões divergentes sobre o comportamento mais adequado para sistema *eMudar*.

Capítulo 7

Considerações Finais

Neste capítulo são apresentadas as conclusões finais a tirar sobre este periodo de estágio na Critical Software.

Durante o estágio, foi possível participar nas três principais tarefas do *SDLC*: Analise Funcional, Desenvolvimento e Garantia de Qualidade, concretizado através do Módulo de Relatórios e do Módulo de Organização e Autorização. Considera-se que a integração com a equipa do projeto do eMudar, permitiu o desenvolvimento de um sentido crítico em relação às necessidades do mercado de trabalho que contrastam com as do meio académico. Salienta-se que nem sempre foi fácil gerir este compromisso entre as necessidades da empresa e as necessidades do meio académico, sobretudo a nível da definição de âmbito e da escrita do relatório. Ainda que trabalhar numa equipa multidisciplinar, possa, por vezes, constituir um desafio, considera-se que este facto introduz uma variabilidade benéfica no *SDLC* contribuindo para maior versatilidade da mesma. Destaca-se também que a relação próxima entre os *SPAEs* e os programadores, contribui para uma interação e comunicação mais eficaz, constituindo uma vantagem para o projeto.

Processos

Considera-se que a conformidade com certificações como o *CMMI* nível 5 contribuem para que a empresa apresente um nível de maturidade de processos muito elevado, sendo a sua aplicação feita de forma cuidada e controlada. Porém, considera-

se que face à inexperiência de um estagiário e ao seu reduzido contexto inicial acerca do projeto, seria desejável que as suas estimativas fossem complementadas formalmente por um "Expert Judgement", de forma a evitar problemas habituais como o otimismo na estimação e a omissão de tarefas como o *refactoring*, *deployment* e estabilização. Apesar disso, considera-se que a utilização de técnicas como o *3-point estimation*, contribuíram positivamente para lidar com a incerteza existente na realização de algumas tarefas. Foi também possível constatar a utilidade que uma boa análise de riscos evidencia, favorecendo a mitigação de situações que pudessem afetar o sucesso do estágio.

Implementação

Durante a fase de codificação, o presente estágio permitiu que fosse feito um desenvolvimento "extremo-a-extremo", ao englobar as diversas camadas do sistema, começando pelo esquema de dados, evoluindo para a camada de serviços e terminando na interface ao utilizador. Sendo o *eMudar* um sistema *business-critical*, a qualidade da implementação é um atributo altamente necessário, pelo que se proporcionou o estudo e a aprendizagem de múltiplos *design-patterns* ao longo das diversas componentes em que o estagiário esteve envolvido.

Garantia de Qualidade de Software

No que diz respeito à garantia de qualidade, este estágio favoreceu que se obtivesse um melhor entendimento acerca da importância e da complexidade desta área, sendo um desafio conciliar o tempo limitado para testes com o extenso universo de técnicas que podem ser utilizadas. A nível da criação de uma suite de testes automatizados, observou-se que a principal dificuldade não está na automatização dos testes em si, mas em garantir a fiabilidade, robustez e manutenibilidade da mesma, sobretudo atendendo que assenta em código que não é entregue ao cliente.

Para concluir, destaca-se que tanto os objetivos principais do estágio, como os opcionais, foram atingidos com sucesso, tendo a metodologia seguida sido eficaz para lidar com a complexidade do projeto *eMudar* e com os desafios que foram surgindo, apesar da existência de alguns desvios face ao planeamento inicial.

Referências

[1] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2 edition, 2003. xiii, 13, 14

[2] Safira. Business process management - bpm. `http://www.safira.pt/pt/pt/paginas/114/business-process-management-bpm.html`. Acesso em: Janeiro, 2016. xv

[3] Margaret Rouse. Definition customer relationship management (crm). `http://searchcrm.techtarget.com/definition/CRM`, Novembro 2014. xv

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. xvi, 59, 62, 65

[5] Microsoft Patterns & Practices Team. *Application Architecture Guide (Patterns & Practices)*. Microsoft Press, second edition, Novembro 2009. xvi, 62

[6] Microsoft. Chapter 1: Service oriented architecture (soa). `https://msdn.microsoft.com/en-us/library/bb833022.aspx`. Acesso em Fevereiro, 2016. xvii

[7] Tim Bray, François Yergeau, Eve Maler, Jean Paoli, and Michael Sperberg-McQueen. Extensible markup language (XML) 1.0 (fifth edition). W3C recommendation, W3C, Novembro 2008. http://www.w3.org/TR/2008/REC-xml-20081126/. xvii

[8] Margaret Rouse and Linda Tucci. Information age. http://searchcio.techtarget.com/definition/Information-Age, Março 2014. 1

[9] CMMI Institute. What is cmmi? http://cmmiinstitute.com/what-is-cmmi. Acesso em: Janeiro, 2016. 2

[10] Microsoft. Background to cmmi. https://msdn.microsoft.com/en-us/library/ee461556.aspx, 2015. Acesso em: Janeiro, 2016. 2

[11] BFA. Bfa@glance. http://www.bfa.ao/Conteudos/Medias/Download.aspx?idi=10033, Dezembro 2015. Acesso em: Janeiro, 2016. 2, 3

[12] Agência Lusa. Bpi vai analisar proposta de compra de 10% do banco angolano pela unitel. http://economico.sapo.pt/noticias/bpi-vai-analisar-proposta-de-compra-de-10-do-banco-angolano-pela-unitel_238941.html, Janeiro 2016. Acesso em: Janeiro, 2016. 2

[13] Eric Kimberling. Five reasons why business process reengineering should happen before your erp implementation. http://goo.gl/MDXkRE, Outubro 2012. 3

[14] Jon Roskill. What comes first: Business process reengineering or erp implementation? http://www.acumatica.com/what-comes-first-business-process-reengineering-or-erp-implementation/, Julho 2014. 3

[15] Critical Software. Connected processes for better banking. http://info.criticalsoftware.com/connected-processes-for-better-banking-lp, 2015. Acesso em: Dezembro, 2015. 3

[16] BFA. Relatório e contas'14. http://www.bfa.ao/Conteudos/Medias/Download.aspx?idi=9512, Março 2015. 4

[17] Gregory Tassey. The economic impacts of inadequate infrastructure for software testing. Planning report, National Institute of Standards and Technology (NIST), Maio 2002. Acesso: http://www.nist.gov/director/planning/upload/report02-3.pdf. 7, 84

[18] Julie Cohen, Dan Plakosh, and Kristi Keeler. Robustness testing of software-intensive systems: Explanation and guide. Technical report, Software Engineering Institute (SEI) - Carnegie Mellon University (CMU), Abril 2005. 8, 84, 111

[19] redhat. Community or enterprise? choosing between jboss community projects and red hat jboss middleware. https://goo.gl/k881Ir, Maio 2015. Whitepaper. 17

[20] Oracle. Java ee compatibility. http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html. Acesso em Dezembro, 2015. 17, 18

[21] Apache. Apache geronimo service and support. http://geronimo.apache.org/service-and-support.html. Acesso em Novembro, 2015. 17

[22] IBM. Ibm withdrawal announcement. http://www-01.ibm.com/common/ssi/rep_ca/1/897/ENUS913-081/ENUS913-081.PDF, Maio 2014. 17

[23] Roman Kharkovski. Websphere app server vs jboss vs weblogic vs tomcat. http://pt.slideshare.net/Romanik/websphere-app-server-vs-jboss-vs-weblogic-vs-tomcat, Fevereiro 2015. Acesso em Novembro, 2015. 18

[24] Gartner. Worldwide application infrastructure and middleware market 2014. http://www.gartner.com/newsroom/id/3034519, Abril 2015. 18

[25] Oliver White. Java tools & technologies landscape for 2014. http://pages.zeroturnaround.com/Java-Tools-Technologies.html, Maio 2014. 18

[26] Arjan Tijms. Diving into the unknown: the jeus application server. http://arjan-tijms.omnifaces.org/2013/09/diving-into-unknown-jeus-application.html, Setembro 2013. 18

[27] Roman Kharkovski. Websphere app server vs jboss vs weblogic vs tomcat (interconnect 2016). http://slideshare.net/Romanik/websphere-app-server-vs-jboss-vs-weblogic-vs-tomcat-interconnect-2016, Fevereiro 2016. Slides 10-18. 19

[28] Redwood Consulting Group. Cloud application infrastructure pricing. https://www.vmware.com/files/pdf/Redwood_White_Paper-Cloud_Application_Infrastructure_Pricing_-_September_2012.pdf, Setembro 2012. 19

[29] Gowri Shankar Palani. Summary of web application testing methodologies and tools. Technical report, IBM, Março 2011. http://www.ibm.com/developerworks/library/wa-webapptesting/wa-webapptesting-pdf.pdf. 21, 22

[30] Mark Buenen, Ajay Walgude, Ngaire Mckeown, and Archit Revandkar. World quality report 2015-16. Technical Report 7ª edição, Capgemini and HP and Sogeti, Setembro 2015. http://techbeacon.com/sites/default/files/gated_asset/world-quality-report-2015-16_v15.pdf. 21

[31] Cem Kaner. Architectures of test automation. Technical report, Department of Computer Science Florida Institute of Technology, Outubro 2000. Acesso: http://kaner.com/pdfs/testarch.pdf. 21, 90, 123

[32] Maritess Sobejana Thomas E. Murphy, Joachim Herschmann. Magic quadrant for software test automation. Technical report, Gartner, Dezembro 2015. https://www.gartner.com/doc/reprints?id=1-2TXA7SZ&ct=151211. 22, 27

[33] Vijay. Most popular web application testing tools – comprehensive list. http://www.softwaretestinghelp.com/most-popular-web-application-testing-tools, Março 2016. 22

[34] Alister Scott. Real vs headless browsers for automated acceptance tests. https://watirmelon.com/2015/12/08/real-vs-headless-browsers-for-automated-acceptance-tests/, Dezembro 2015. 22

[35] Atif M. Memon. GUI testing: Pitfalls and process. *Computer*, 35(8):87–88, Agosto 2002. http://www-lb.cs.umd.edu/~atif/papers/MemonIEEEComputer2002.pdf. 24, 85

[36] Gerard Meszaros. *XUnit Test Patterns: Refactoring Test Code*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006. 24, 26, 76, 79, 92, 93, 111, 120

[37] Martin Fowler. Xunit. http://www.martinfowler.com/bliki/Xunit.html, Janeiro 2006. 25, 28

[38] Gerard Meszaros, Shaun M. Smith, and Jennitta Andrea. *The Test Automation Manifesto*, pages 73–81. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. 26, 59, 65, 92, 93

[39] Tutorials Point. Sdlc - waterfall model. http://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm. Acesso em Dezembro, 2015. 34

[40] Tutorials Point. Sdlc - v-model. http://www.tutorialspoint.com/sdlc/sdlc_v_model.htm. Acesso em Dezembro, 2015. 35

[41] International Software Testing Qualifications Board (ISTQB). Glossário standard de termos usados em testes de software. http://media.wix.com/ugd/928ecf_fb7a9676daae44ea9b1572a2a12c53ea.pdf, Outubro 2012. v2.2pt. 36, 37, 76, 78, 83, 112

[42] Christopher Alberts, Audrey Dorofee, Ron Higuera, Richard Murphy, Julie Walker, and Ray Williams. *Continuous Risk Management Guidebook*. Software Engineering Institute, Carnegie Mellon University, 1996. 36, 37

[43] Liam O'Brien, Len Bass, and Paulo Merson. Quality attributes and service-oriented architectures. Technical Report CMU/SEI-2005-TN-014, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2005. 46

[44] Robert L. Glass. Frequently forgotten fundamental facts about software engineering. *IEEE Software*, 18(3):112–111, Maio 2001. 46, 64, 68, 90

[45] Felix Bachmann, Len Bass, and Robert Nord. Modifiability tactics. Technical Report CMU/SEI-2007-TR-002, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2007. 47, 51, 59, 62

[46] Felix Bachmann, Len Bass, Paul Clements, David Garlan, James Ivers, M. Little, Paulo Merson, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, second edition, 2010. 51

[47] Alex Homer, John Sharp, Larry Brader, Masashi Narumoto, and Trent Swanson. *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*. Microsoft patterns & practices, 2014. 53, 60

[48] Matt Allen. Relational databases are not designed for heterogeneous data. http://www.marklogic.com/blog/relational-databases-heterogeneous-data, Novembro 2015. 53

[49] Pramod J. Sadalage and Martin Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 1st edition, 2012. 54

[50] John Snelson, Don Chamberlin, Michael Dyck, and Jonathan Robie. XML path language (XPath) 3.0. W3C recommendation, W3C, Abril 2014. http://www.w3.org/TR/2014/REC-xpath-30-20140408/. 55, 59, 60

[51] Jonathan Robie, John Snelson, Michael Dyck, and Don Chamberlin. XQuery 3.0: An XML query language. W3C recommendation, W3C, Abril 2014. http://www.w3.org/TR/2014/REC-xquery-30-20140408/. 55

[52] Cynthia M. Saracco. Get off to a fast start with db2 9 purexml, part 1. Technical report, IBM, Março 2010. http://www.ibm.com/developerworks/data/library/techarticle/dm-0602saracco/dm-0602saracco-pdf.pdf. 55

[53] Cynthia M. Saracco. Get off to a fast start with db2 9 purexml, part 2:. Technical report, IBM, Março 2010. http://www.ibm.com/developerworks/data/library/techarticle/dm-0603saracco/dm-0603saracco-pdf.pdf. 55

[54] Cynthia M. Saracco. Get off to a fast start with db2 9 purexml, part 3:. Technical report, IBM, Março 2010. http://www.ibm.com/developerworks/data/library/techarticle/dm-0603saracco2/dm-0603saracco2-pdf.pdf. 55

[55] William F. Opdyke. *Refactoring Object-oriented Frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1992. UMI Order No. GAX93-05645. 59

[56] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, first edition, 1999. 59, 65, 67, 69, 70

[57] Open Web Application Security Project (OWASP). The 10 most critical web application security risks. http://goo.gl/oTCDS, 2013. 61, 103

[58] Ethan Marcotte. Responsive web design. Maio 2010. http://alistapart.com/article/responsive-web-design. 66

[59] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. 67, 74

[60] George Fairbanks and David Garlan. *Just Enough Software Architecture: A Risk-driven Approach*. Marshall & Brainerd, 2010. 69

[61] Alessandro Nadalin. On monoliths, service-oriented architectures and microservices. http://odino.org/on-monoliths-service-oriented-architectures-and-microservices, Fevereiro 2015. 69

[62] Simon Brown. Distributed big balls of mud. http://www.codingthearchitecture.com/2014/07/06/distributed_big_balls_of_mud.html, Julho 2014. 69

[63] Markus Clermont. Automating tests vs. test-automation. http://googletesting.blogspot.pt/2007/10/automating-tests-vs-test-automation.html, Outubro 2007. 76, 79

[64] Kari Briski, Poonam Chitale, Valerie Hamilton, Allan Pratt, Brian Starr, Jim Veroulis, and Bruce Villard. Minimizing code defects to improve software quality and lower development costs. White paper, IBM, Outubro 2008. ftp://ftp.software.ibm.com/software/rational/info/do-more/RAW14109USEN.pdf. 84

[65] Nuno Laranjeiro. Testing the robustness of services. http://www.critical-step.eu/index.php/meeting-documents/doc_download/122-testing-the-robustness-of-services, Novembro 2011. 85

[66] David J. Coe. A review of boundary value analysis techniques. http://static1.1.sqspcdn.com/static/f/702523/9242142/1288741669800/200804-Coe.pdf, Abril 2008. 86

[67] Philip Koopman, Kobey DeVale, and John DeVale. Interface robustness testing: Experience and lessons learned from the ballista project. *Dependability Benchmarking for Computer Systems*, 72:201, 2008. 88

[68] Nuno Laranjeiro, Marco Vieira, and Henrique Madeira. A robustness testing approach for SOAP web services. *J. Internet Services and Applications*, 3(2):215–232, 2012. 88

[69] Dale H. Emery. Writing maitainable automated acceptance tests. http://dhemery.com/pdf/writing_maintainable_automated_acceptance_tests.pdf, 2009. 90, 92

[70] Dawn Haynes. Automated testing: A silver bullet? Technical report, Rational Software (IBM), 2001. Acesso: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jun01/AutomatedTestingJune01.pdf. 90

[71] Martin Fowler. Pageobject. http://martinfowler.com/bliki/PageObject.html, Setembro 2013. 94

[72] Selenium Project. Page object design pattern. http://docs.seleniumhq.org/docs/06_test_design_considerations.jsp#page-object-design-pattern. Acesso em Maio, 2016. 94

[73] Mozilla Developer Network. Deploying firefox in an enterprise environment. https://developer.mozilla.org/en-US/Firefox/Enterprise_deployment, Fevereiro 2016. 96

[74] Justin Schuh. The final countdown for npapi. http://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html, Novembro 2014. 101

[75] Chris Hoffman. Why browser plug-ins are going away and what's replacing them. http://www.howtogeek.com/179213/why-browser-plug-ins-are-going-away-and-whats-replacing-them, Julho 2014. Acedido em Maio de 2016. 101

[76] Benjamin Smedberg. Npapi plugins in firefox. https://blog.mozilla.org/futurereleases/2015/10/08/npapi-plugins-in-firefox, Outubro 2015. 101

[77] Dalibor Topic. Migrating from java applets to plugin-free java technologies. White paper, Oracle, Janeiro 2016. http://www.oracle.com/technetwork/java/javase/migratingfromapplets-2872444.pdf. 101

[78] Petko Petkov, Adrian Pastor, David Kierznowski, Mario Heiderich, and Ivana Kalay. Csrf demystified. www.gnucitizen.org/blog/csrf-demystified#CountermeasuresandprotectionagainstCSRF, Novembro 2007. 103

[79] Dave Wichers, Paul Petefish, and Eric Sheridan. Cross-site request forgery (csrf) prevention cheat sheet. https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet, Maio 2016. 103

[80] ZK. Security tips - cross-site request forgery. https://www.zkoss.org/wiki/ZK%20Developer's%20Reference/Security%20Tips/Cross-site%20Request%20Forgery, Setembro 2014. 103

[81] Zac Campbell. Writing reliable locators for selenium and webdriver tests. https://blog.mozilla.org/webqa/2013/09/26/writing-reliable-locators-for-selenium-and-webdriver-tests, Setembro 2013. 103

[82] Mark Davis. Unicode over 60 percent of the web. https://googleblog.blogspot.pt/2012/02/unicode-over-60-percent-of-web.html, Fevereiro 2012. 106

[83] Ned Batchelder. Pragmatic unicode. http://nedbatchelder.com/text/unipain.html, Março 2012. 107

[84] Diego Garcia. Svn e jenkins: Integração contínua. http://www.devmedia.com.br/svn-e-jenkins-integracao-continua/30833. Acedido em Maio de 2016. 109

[85] Nick Rutar, Christian B. Almazan, and Jeffrey S. Foster. A comparison of bug finding tools for java. In *Proceedings of the 15th International Symposium on Software Reliability Engineering*, ISSRE '04, pages 245–256, Washington, DC, USA, 2004. IEEE Computer Society. 112

[86] Steve Easterbrook. What are requirements. http://www.cs.toronto.edu/~sme/papers/2004/FoRE-chapter02-v7.pdf, 2004. 120

[87] Laura Brandenburg. How to manage change requests. http://www.bridging-the-gap.com/how-to-manage-change-requests. Acesso em Julho, 2016. 123

Apêndice A

Diagramas de Gantt

- Na página 138 encontra-se o resumo das atividades desenvolvidas durante 1 semestre.
- Na página 139 encontra-se o planeamento inicial efetuado para o 2º Semestre.
- Na página 140 encontra-se o resumo das atividades desenvolvidas ao longo do 2º Semestre.
- Na página 141 encontra-se o planeamento/resumo das atividades desenvolvidas na 3ª fase do estágio (período compreendido entre 20 de Junho e 31 de Agosto de 2016).

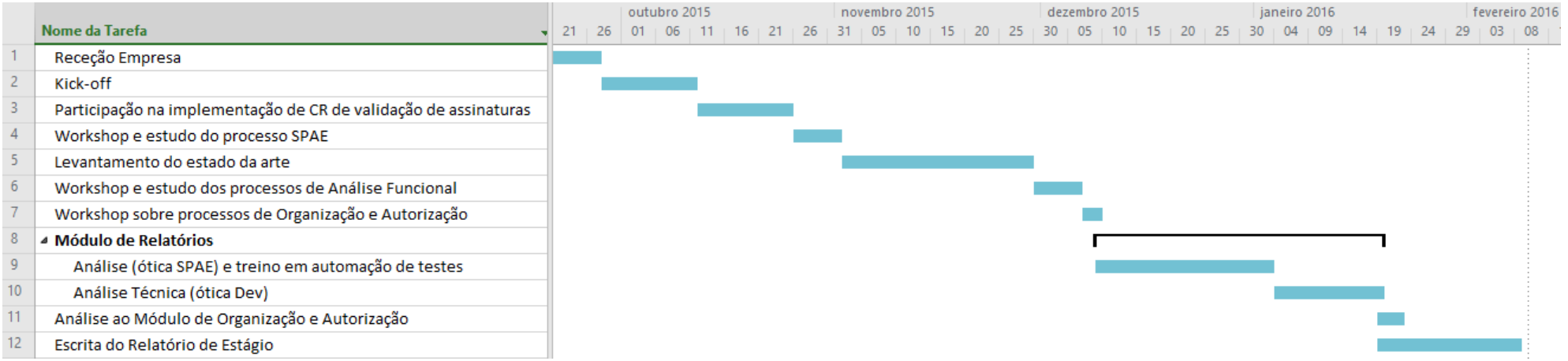
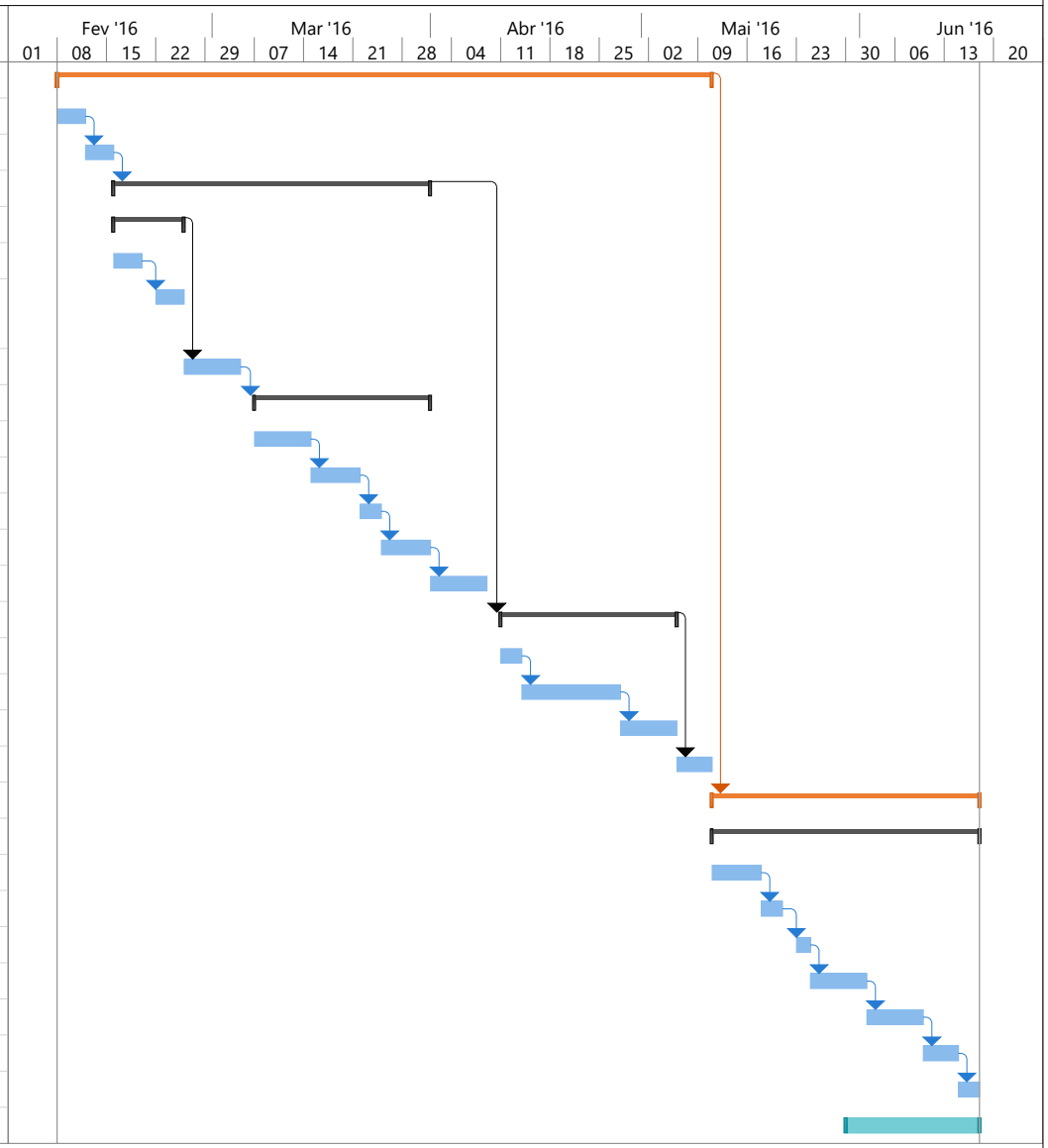


Figura A.1: Diagrama de Gantt relativo às atividades desenvolvidas ao longo do primeiro semestre

Planeamento inicial para o 2º Semestre

ID	Nome da Tarefa	Duração	Início
1	**Módulo de Relatórios**	**67 dias**	**08-02-16**
2	Formalização da Especificação de Testes de Software*	4 dias	08-02-16
3	Familiarização com a framework ZK para web development	2 dias	12-02-16
4	**Implementação**	**33 dias**	**16-02-16**
5	**Cálculo de estatísticas agregadas**	**8 dias**	**16-02-16**
6	Criação da camada de persistência de dados	4 dias	16-02-16
7	Criação de aplicação que proceda à execução de rotinas de cálculo e armazenamento dos dados	4 dias	22-02-16
8	Criação de Web Service para o acesso aos dados	6 dias	26-02-16
9	**Camada de Apresentação**	**19 dias**	**07-03-16**
10	Implementação de Ecrã Genérico	6 dias	07-03-16
11	Implementação dos 8 ecrãs que instanciam o ecrã genérico	5 dias	15-03-16
12	Implementação do ecrã de Monitorização de Órgãos	3 dias	22-03-16
13	Implementação do ecrã de Interação com Processos	5 dias	25-03-16
14	Documentação Relatório de Estágio	6 dias	01-04-16
15	**Garantia de Qualidade**	**19 dias**	**11-04-16**
16	Conclusão do treino em automação de testes**	3 dias	11-04-16
17	Implementação dos Testes de Aceitação	10 dias	14-04-16
18	Documentação Relatório de Estágio	6 dias	28-04-16
19	Correção de Defeitos	3 dias	06-05-16
20	**Módulo de Organização e Autorização**	**28 dias**	**11-05-16**
21	**Análise Funcional**	**28 dias**	**11-05-16**
22	Análise do módulo e dos pedidos de alteração do cliente	5 dias	11-05-16
23	Prototipagem	3 dias	18-05-16
24	*Storyboard*	2 dias	23-05-16
25	Documentação Relatório de Estágio	6 dias	25-05-16
26	Atualização da especificação de Requisitos	6 dias	02-06-16
27	Compilação de resumo de alterações para implementação	3 dias	10-06-16
28	Correções à especificação proposta	3 dias	15-06-16
29	Finalização do Relatório de Estágio	15 dias	30-05-16

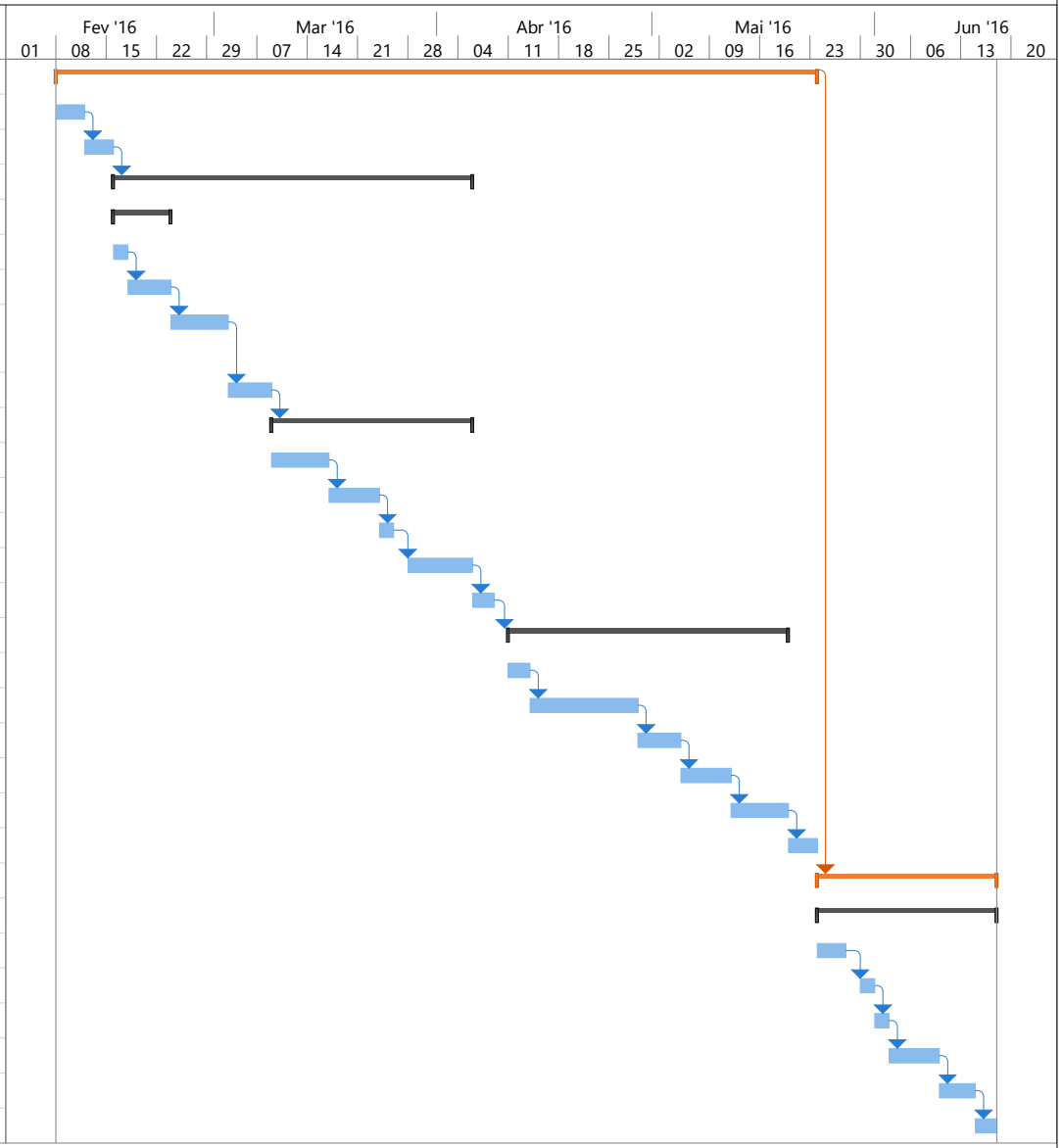


* A especificação de testes foi iniciada no 1º semestre, sendo concluída e formalizada num documento no âmbito desta tarefa

** O Treino em automação de testes foi iniciado no 1º semestre

Resumo das atividades do 2º Semestre - Parte 1

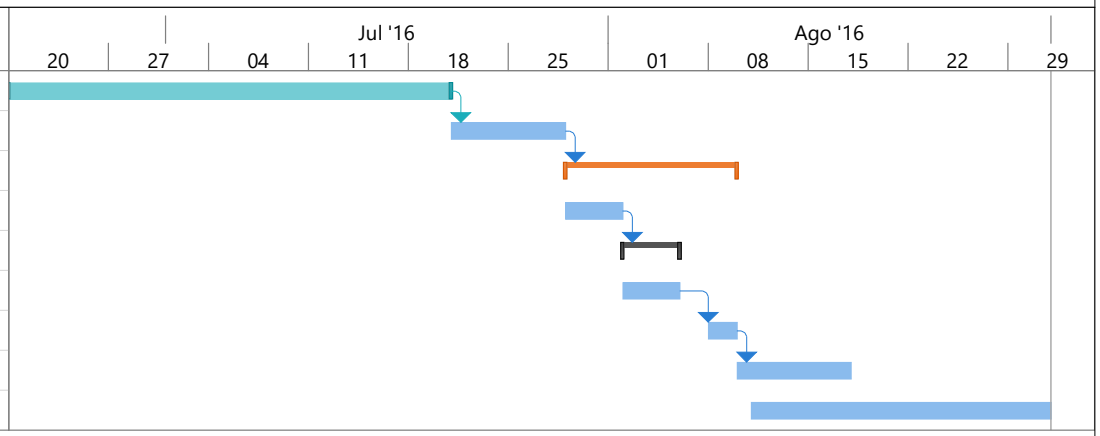
ID	Nome da Tarefa	Duração	Início
1	**Módulo de Relatórios**	**76 dias**	**08-02-16**
2	Formalização da Especificação de Testes de Software*	4 dias	08-02-16
3	Familiarização com a framework ZK para web development	2 dias	12-02-16
4	**Implementação**	**36 dias**	**16-02-16**
5	**Conceção do Modelo de Dados**	**6 dias**	**16-02-16**
6	Análise e estudo dos mecanismos obtenção de dados	2 dias	16-02-16
7	Criação da camada de persistência de dados	4 dias	18-02-16
8	Criação de aplicação que proceda à execução de rotinas de cálculo e armazenamento dos dados	6 dias	24-02-16
9	Criação de Web Service para o acesso aos dados	4 dias	03-03-16
10	**Camada de Apresentação**	**20 dias**	**09-03-16**
11	Implementação de Ecrã Genérico	6 dias	09-03-16
12	Implementação dos 8 ecrãs que instanciam o ecrã genérico	5 dias	17-03-16
13	Implementação do ecrã de Monitorização de Órgãos	2 dias	24-03-16
14	Implementação do ecrã de Interação com Processos	7 dias	28-03-16
15	Documentação Relatório de Estágio	3 dias	06-04-16
16	**Garantia de Qualidade**	**29 dias**	**11-04-16**
17	Conclusão do treino em automação de testes**	3 dias	11-04-16
18	Implementação dos Testes de Aceitação	11 dias	14-04-16
19	Estabilização e Testes de Fiabilidade à Suite	4 dias	29-04-16
20	Documentação Relatório de Estágio	5 dias	05-05-16
21	Início dos Testes de Robustez	6 dias	12-05-16
22	Correção de Defeitos	2 dias	20-05-16
23	**Módulo de Organização e Autorização**	**19 dias**	**24-05-16**
24	**Análise Funcional**	**19 dias**	**24-05-16**
25	Análise do módulo e dos pedidos de alteração do cliente	4 dias	24-05-16
26	Prototipagem	2 dias	30-05-16
27	*Storyboard*	2 dias	01-06-16
28	Atualização da especificação de Requisitos	5 dias	03-06-16
29	Compilação de resumo de alterações para implementação	3 dias	10-06-16
30	Correções à especificação de requisitos proposta	3 dias	15-06-16



* A especificação de testes foi iniciada no 1º semestre, sendo concluída e formalizada num documento no âmbito desta tarefa
** O Treino em automação de testes foi iniciado no 1º semestre

Resumo das atividades do 2º Semestre - Parte 2

ID	Nome da Tarefa	Duração	Início	Conclusão
1	Interrupção do Estágio	23 dias	20-06-16	20-07-16
2	Documentação Relatório de Estágio	6 dias	21-07-16	28-07-16
3	**Módulo de Relatórios**	**8 dias**	**29-07-16**	**09-08-16**
4	Preparativos para deployment	2 dias	29-07-16	01-08-16
5	**Garantia de Qualidade**	**4 dias**	**02-08-16**	**05-08-16**
6	Conclusão dos Testes de Robustez	4 dias	02-08-16	05-08-16
7	Correção de Defeitos	2 dias	08-08-16	09-08-16
8	Documentação Relatório de Estágio	6 dias	10-08-16	17-08-16
9	Finalização do Relatório de Estágio	15 dias	11-08-16	31-08-16



Apêndice B

Especificação de Requisitos de Software do Módulo de Relatórios

Apêndice ***confidencial*** incluído apenas no CD.

Relembra-se que este documento foi produzido pela equipa de análise funcional do projeto, tendo o estagiário participado apenas na sua revisão no âmbito da inspeção de requisitos detalhado na secção 5.2.

B. ESPECIFICAÇÃO DE REQUISITOS DE SOFTWARE DO MÓDULO DE RELATÓRIOS

Apêndice C

Storyboard do Módulo de Relatórios

Apêndice ***confidencial*** incluído apenas no CD.

Apêndice D

Diagramas de Classes relativos às rotinas do Módulo de Relatórios

- Na página 148 encontra-se representado um *Diagrama de Classes* com as principais classes desenvolvidas no âmbito das rotinas do Módulo de Relatórios do *eMudar*.
- Na página 149 encontra-se representado um Diagrama de Classes com vista a destacar as capacidades de extensibilidade e modificabilidade de relatórios que interagem com a *BPMS* .

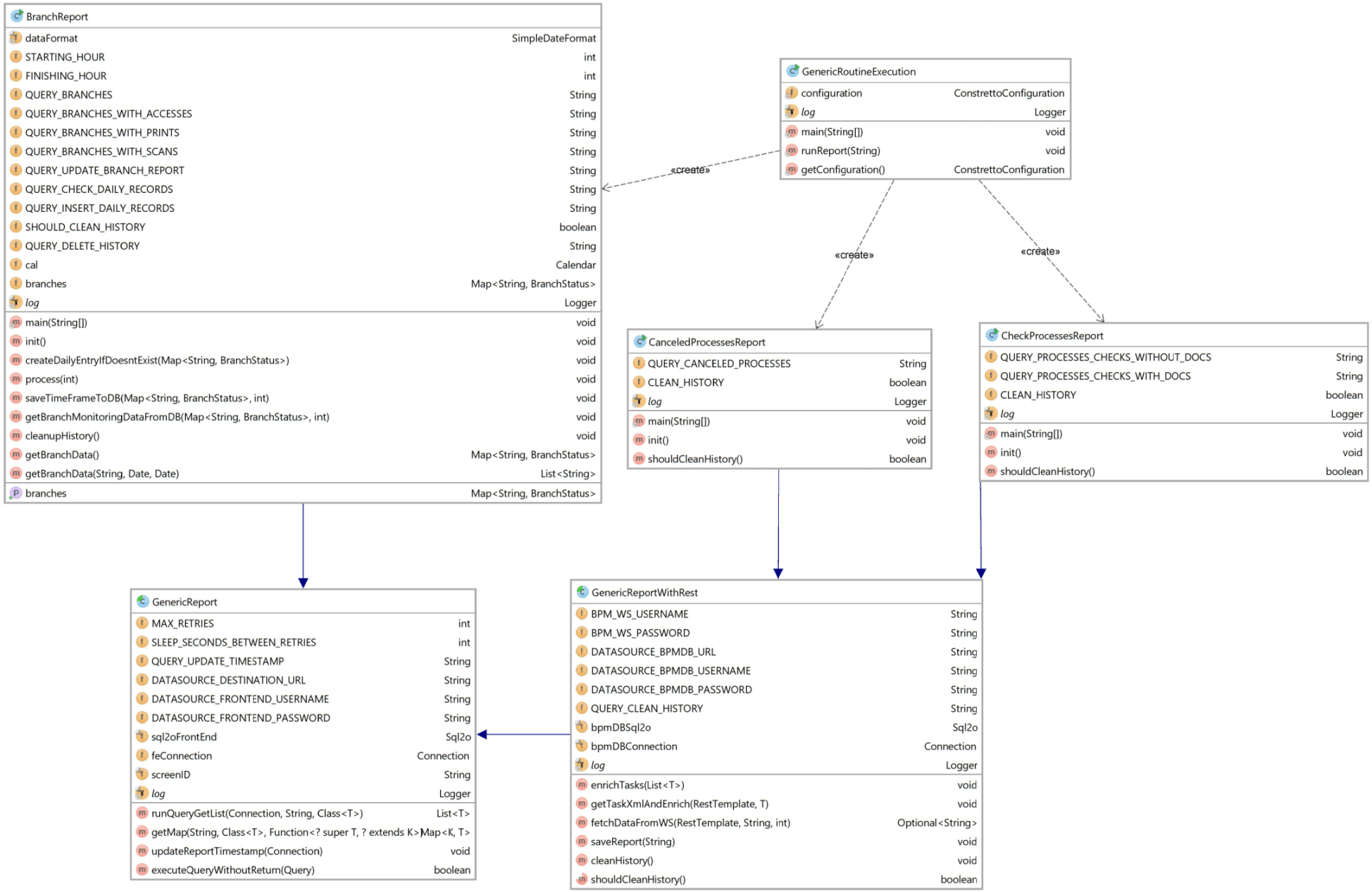
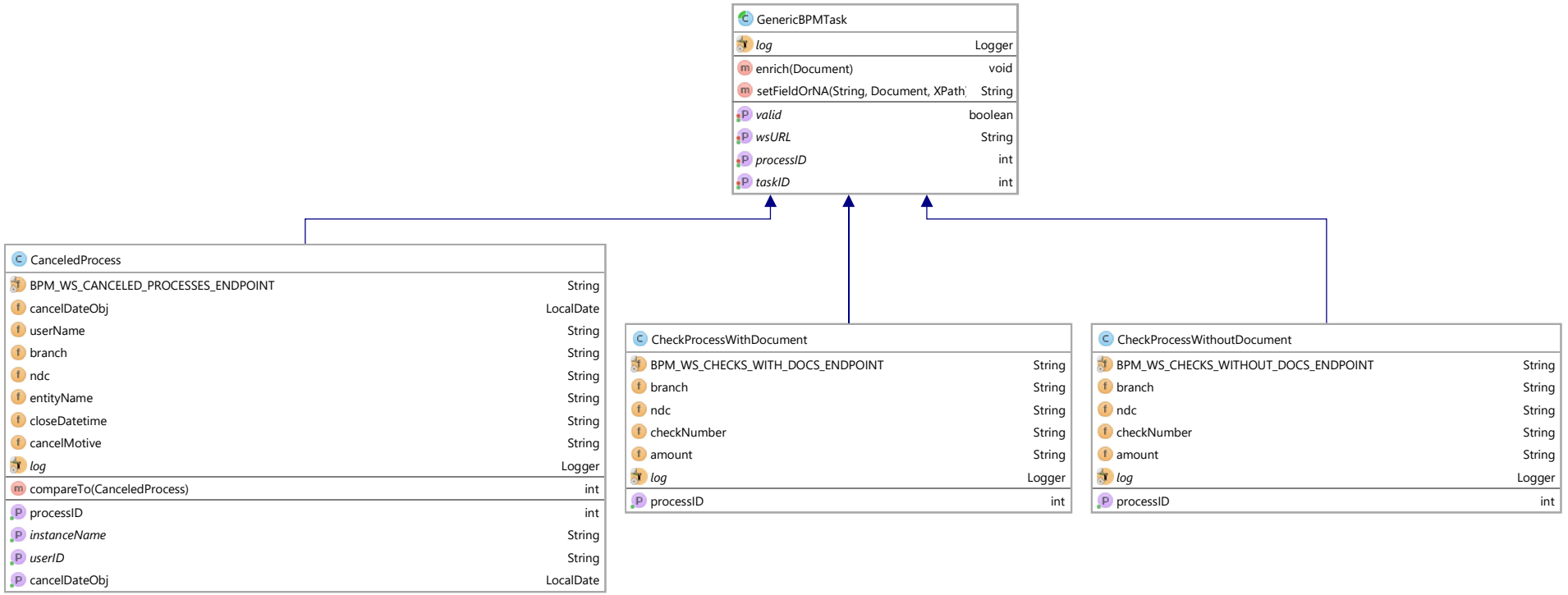


Figura D.1: Diagrama de Classes UML com as principais classes desenvolvidas no âmbito das rotinas do Módulo de Relatórios


GenericBPMTask
log Logger
enrich(Document) void
setFieldOrNA(String, Document, XPath) String
valid boolean
wsURL String
processID int
taskID int
CanceledProcess
BPM_WS_CANCELED_PROCESSES_ENDPOINT String
cancelDateObj LocalDate
userName String
branch String
ndc String
entityName String
closeDatetime String
cancelMotive String
log Logger
compareTo(CanceledProcess) int
processID int
instanceName String
userID String
cancelDateObj LocalDate
CheckProcessWithDocument
BPM_WS_CHECKS_WITH_DOCS_ENDPOINT String
branch String
ndc String
checkNumber String
amount String
log Logger
processID int
CheckProcessWithoutDocument
BPM_WS_CHECKS_WITHOUT_DOCS_ENDPOINT String
branch String
ndc String
checkNumber String
amount String
log Logger
processID int

D. DIAGRAMAS DE CLASSES RELATIVOS ÀS ROTINAS DO MÓDULO DE RELATÓRIOS

Apêndice E

Plano de Testes do Módulo de Relatórios

Apêndice ***confidencial*** incluído apenas no CD.

Apêndice F

Storyboard efetuado para o Módulo de Organização e Autorização

Apêndice ***confidencial*** incluído apenas no CD.

Apêndice G

Especificação de Requisitos de Software do Módulo de Organização e Autorização

Apêndice ***confidencial*** incluído apenas no CD.