INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Plataforma de apoio ao processo de Software Release Planning

LUÍS MIGUEL PEREIRA AUGUSTO
Junho de 2021

POLITÉCNICO
DO PORTO

Plataforma de apoio ao processo de Software Release Planning

Luís Miguel Pereira Augusto

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Alexandre Bragança
Supervisor: Ana Matos

Júri:
Presidente:

Vogais:

Porto, 30 de junho de 2021

Resumo

Com a adopção de metodologias ágeis para o desenvolvimento de *software* e a necessidade cada vez mais urgente de acrescentar mais funcionalidades em menor tempo aos *softwares*, o processo de *release planning* tem desempenhado um papel de grande importância nas empresas que desenvolvem *software*.

ALERT Life Sciences Computing, S.A. é uma empresa que desenvolve *software* para a área dos cuidados médicos e, como tal, o planeamento do desenvolvimento de *software* é de extrema importância.

Esta dissertação relata o trabalho que foi desenvolvido para pensar e construir uma ferramenta que apoie o processo de *release planning* da ALERT Life Sciences Computing, S.A.

Ao longo do documento é descrito o processo de desenho e implementação de um *software* capaz de ajudar a equipa de *release planning* a planear cada *release* do produto ALERT ®EMR.

No final foi avaliado o desempenho da solução. Apesar de, ser uma plataforma que se prevê em constante crescimento, o produto de *software* atual é satisfatório.

Palavras-chave: Release Planning, Desenvolvimento Web, Integração com o Jira, Schedule Network Analysis

Abstract

With the adoption of agile for software development and the increasing need to add more functionality in less time to software, the release planning process has played a very important role in software development companies.

ALERT Life Sciences Computing, S.A. is a company that develops healthcare software, therefore, software development planning is of extreme importance.

This dissertation reports on the work that was developed to design and build a tool that supports the release planning process at ALERT Life Sciences Computing, S.A.

Throughout the document, is described the design and implementation process of a software tool capable of helping the release planning team to plan each release of the ALERT ®EMR.

At the end the solution's performance was evaluated. Despite being a platform that is expected to grow constantly, the current software product is satisfactory.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Código

Lista de Acrónimos

AD	Active Directory.
AHP	Analytic Hierarchy Process.
API	Application Programming Interface.
CCM	Critical Chain Method.
CPM	Critical Path Method.
DB	Data Base.
DDL	Data Definition Language.
DTO	Data Transfer Object.
FFE	Fuzzy Front End.
GA	Genetic Algorithm.
HMO	Healthcare Management Organizations.
HTTP	Hypertext Transfer Protocol.
IC	Índice de Consistência.
ILP	Integer Linear Programming.
IR	Índice Randómico.
ISEP	Instituto Superior de Engenharia do Porto.
JQL	Jira Query Language.
JSON	JavaScript Object Notation.
LDAP	Lightweight Directory Access Protocol.
MEI	Mestrado em Engenharia Informática.
MW	Middleware.
NCD	New Concept Developmen.
NPD	New Product Development.
NPPD	New Product and Process Development.
PERT	Program Evaluation and Review Technique.
PL/SQL	Procedural Language/Structured Query Language.
RC	Razão de Consistência.

RCPSP	Resource Constrained Project Scheduling Problem.
REST	Representational State Transfer.
SDD	Short Sequence Diagram.
SPA	Single-Page Application.
SQL	Structured Query Language.
SVN	Subversion.
TMDEI	Tese/Estágio/Dissertação.
UI/UX	User Interface/User experience.
UML	Unified Modeling Language.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.

Capítulo 1

Introdução

Este é o capítulo introdutório desta dissertação e descreve o projeto realizado no âmbito da unidade curricular Tese/Estágio/Dissertação (TMDEI) do Mestrado em Engenharia Informática (MEI) – ramo de especialização Engenharia de Software - do Instituto Superior de Engenharia do Porto (ISEP). Neste capítulo, visa-se apresentar uma breve descrição sobre em que contexto se insere o projeto, o problema que o despoletou e quais objetivos que se pretende alcançar. Além disto, é também, apresentada a metodologia de trabalho adotada para atingir os objetivos definidos e a estrutura do documento.

1.1 Contexto

Actualmente, o desenvolvimento de *software* segue, tendencialmente abordagens ágeis em detrimento dos modelos *waterfall*. Sendo o desenvolvimento iterativo e incremental considerado um aspecto fundamental para as abordagens ágeis [1].

O desenvolvimento incremental permite que os consumidores de um *software* recebam, de forma periódica, partes de um sistema. Isto, proporciona a criação de valor antecipado de um *software* bem como a recepção de *feedback* constante de quem o utiliza. A cada lançamento do *software* um conjunto de funcionalidades novas são acrescentadas ao produto, o que proporciona ao cliente um sistema completo e valioso. Para além disto, novas *releases* servem, também, para corrigir erros e *bugs* que foram detetatos em versões antigas do produto [2]. Associado a este processo, surgiram os conceitos de *Release Planning* e versão de *software* [1].

Release Planning corresponde à selecção e escalonamento das tarefas necessárias a realizar para a próxima versão do produto de *software*. Estes planos são importantes porque definem como é que uma empresa de *software* equilibra as suas capacidades para a realização de metas a curto prazo e investimentos a longo prazo, adicionando, alterando e removendo ou realocando os seus recursos. Tipicamente, os *Relaese Plans* são realizados em intervalos regulares ou em momentos importantes para a organização do desenvolvimento do produto de *software* [3].

Más decisões durante o planeamento de uma *release* podem resultar em [2]:

- clientes insatisfeitos que não conseguem o que esperam;
- planos improváveis de serem cumpridos dentro de um determinado cronograma, restrições de qualidade e esforço não sendo atendidas;

[1] Nome ou número único de um *software*

- planos que não oferecem o melhor valor comercial para o investimento.

Este projeto, cujo tema é "Plataforma de apoio ao processo de *Software Release Planning*" surge mediante uma proposta apresentada pela empresa ALERT Life Sciences Computing, S.A., doravante mencionada apenas como "ALERT", e visa apresentar uma plataforma que, como o título indica, apoie o processo de *Software Release Planning* da ALERT.

A ALERT foi fundada em 1999 e tem como foco a criação de uma solução integrada para a total informatização dos cuidados de saúde em países inteiros, *Healthcare Management Organizations (HMO)*, hospitais e centros de saúde, passando pelos processos individuais de cada paciente.

Além disto, a ALERT oferece o *software* ALERT ® (Figura 1.1), uma *suite* desenhada e desenvolvida de forma a fornecer uma solução digital clínica completa e independente do ambiente de saúde onde opera, incluindo todos os perfis dos profissionais de saúde, garantindo a partilha de informação com os pacientes e oferecendo modelos de instalação local e serviços de gestão completos [4].

Figura 1.1: ALERT ® [adaptado de [4]]

Assim sendo, o mercado onde se insere a empresa proponente deste projeto é o mercado da saúde. Este é um mercado extremamente exigente e com uma elevada taxa de evolução em Portugal [5] onde registo clínico electrónico de pacientes desempenha um papel crucial na prestação de cuidados de saúde.

Neste sentido, tendo em conta a exigência do mercado onde está inserida, e de modo a manter um patamar de excelência para o seu produto a ALERT desenvolveu um processo composto por diversas actividades relacionadas com a gestão e planeamento do seu *software*. Este projeto debruça-se sobre esse processo de modo a automatizar algumas actividades críticas do mesmo.

1.2 Problema

A ALERT é uma *software house*, ou seja, empresa dedicada ao desenvolvimento de *software*, neste caso de saúde. Por este motivo, a ALERT desenvolve directamente, com base

em metodologias ágeis de desenvolvimento de *software*, todas as fases do ciclo de vida (Figura 1.2) da sua *suite* de produtos.

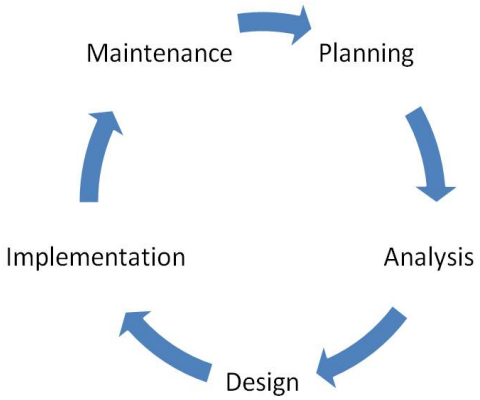

Figura 1.2: Processo de Ciclo de Vida de Desenvolvimento de Software [Fonte: [6]]

A gestão de desenvolvimento do produto é, como já mencionado, de extrema importância. Pelo que, a ALERT deseja uma solução de *software* capaz de se integrar no seu processo de planeamento, automatizando algumas das suas tarefas.

Actualmente, todo o planeamento de *software* é realizado manualmente pela equipa de *Software Release Planning*. A cada novo *Release Plan* a equipa aplica uma série de procedimentos, que resulta num conjunto de requisitos de *software* distribuído pelas diversas equipas de desenvolvimento, isto é, uma série de recursos humanos. Com vista a estimar o final da *release*, a equipa aplica, de forma manual, o método do caminho crítico[2], um tipo de *Schedule Network Analysis*.

A estimativa para o fim de uma *release* é um indicador importante para que a equipa possa efectuar os ajustes necessários ao plano e tem em consideração fatores como: percentagem alocação de recursos humanos à *release*, disponibilidade dos recursos humanos, paralelismo dos desenvolvimentos.

A execução deste processo de forma manual resulta num processo de planeamento moroso e complexo promovendo uma maior probabilidade de falhas, resultantes de falhas humanas.

Dada a importância que a gestão de desenvolvimento do produto representa para a ALERT surgiu a necessidade de criar uma ferramenta capaz de apoiar o planeamento, substituindo as actuais operações manuais por operações automáticas.

1.3 Hipótese

De acordo com o contexto e o problema previamente apresentado, formulou-se a seguinte hipótese: é possível desenhar e desenvolver um *software* capaz de, de forma optimizada,

[2]Algoritmo utilizado para escalonar uma série de tarefas de um determinado projecto [7].

sugerir uma data prevista para o final do desenvolvimento através do cruzamento de dados provenientes da plataforma Jira com os dados de ausências/férias dos diferentes recursos humanos envolvidos no desenvolvimento da *release*. Este *software* deve ser capaz de dar resposta às necessidades da ALERT reduzindo os custos e os riscos associados à actual aplicação do modelo do caminho crítico.

1.4 Objetivos

A plataforma de *software* a ser desenvolvida terá como principal objetivo escalonar, de forma optimizada, as diferentes tarefas associadas ao desenvolvimento de uma *release* com base em dados recolhidos do Jira *Software* e a alocação e disponibilidade dos recursos humanos envolvidos na *release*. Para além disso, a ALERT deseja que a ferramenta reúna dados estatísticos relevantes de cada versão para que possa monitorizar o seu processo de desenvolvimento de *software*.

A plataforma deve estar integrada com o Jira *Software* e permitir que a equipa de *Software Release Planning* consiga definir e redefinir a percentagem de alocação dos recursos humanos à *release*. Deve também ser possível acompanhar, a qualquer momento, o progresso da *release*.

Está prevista a criação dos seguintes módulos de *software*:

- Módulo para gerir os dados relevantes dos recursos humanos;
- Módulo para integração com Jira *Software*;
- Módulo para fazer a gestão do planeamento de uma *release*.

1.5 Contribuições

É esperado que, com este projeto, seja possível à equipa de *Release Planning* manipular de uma forma mais rápida e fácil os dados que influenciam o desenvolvimento de uma *release*. Ou seja, este projeto visa facilitar o trabalho da equipa de *Release Planning* da ALERT de duas formas distintas: através da recolha de dados estatísticos do desenvolvimento de uma plataforma e pela sugestão de uma data para o final do desenvolvimento de uma *release*. Esta sugestão e a facilidade com que irá ser calculada pela plataforma irá permitir à equipa manipular recursos humanos e tarefas que compõe uma *release* de uma forma mais pragmática e simples do que o que até então é praticado pela equipa.

1.6 Análise de valor

A ALERT, deseja obter uma solução relacionada com o planeamento de *software*. O planemento de *software* pode ser dividido em dois: planeamento estratégico e planeamento operacional. Após uma análise sobre o tema, tornou-se evidente que há diversas soluções e propostas de solução para o planeamento estratégico, porém, notou-se um desapreço pelo planeamento operacional. Deste modo, surgiu a necessidade de projetar uma plataforma que se focasse no planeamento operacional.

Este projeto prevê a automatização de processos no decorrer do planeamento do desenvolvimento de *software*, oferecendo um integração com o Jira *Software* e garantindo que agiliza um processo que, actualmente, é manual e moroso. Para além de diminuir o custo

de tempo, propõe reduzir o erro humano aquando o cálculo de uma data estimada para o fim de uma *release*.

O cruzamento da informação recolhida do Jira *Software* com a informação relevante dos recursos humanos permitirá que a equipa de *Software Release Planning* da ALERT consiga planear e acompanhar o planeamento numa única plataforma, facilitando assim o seu trabalho. O aumento da eficiência do cálculo das estimativas e a diminuição do tempo necessário para analisar e planear *releases* providenciará à equipa agilidade no ato de planear e re-planear uma *release*. A descrição completa da análise de valor da solução em questão encontra-se no Capitulo 3.

1.7 Metodologia de trabalho

No início do projeto foi definida uma metodologia de trabalho e um planeamento com os seguintes propósitos: permitir que os objetivos traçados e os requisitos definidos fossem alcançados e aumentar a organização do trabalho a ser desenvolvido. Este projeto foi dividido em duas grandes fases, em concordância com as metas estabelecidas pela unidade curricular onde o projeto está inserido (TMDEI).

Numa primeira fase, após uma integração na equipa, foi realizado um planeamento orientado à investigação necessária para o projeto considerando as seguintes tarefas:

- Estudar e compreender os conceitos associados ao projeto, nomeadamente o conceito de *Release Planning*;
- Investigar quais as soluções existentes e quais são os trabalhos relacionados com o tema;
- Estudar e compreender os conceitos integrantes do Jira *software*, nomeadamente a arquitetura do Jira, a base de dados do Jira, o esquema de dados do Jira, a linguagem de consulta do Jira -*Jira Query Language (JQL)*, entre outros;
- Estudar e avaliar interfaces de integração com o Jira *Software*;
- Analisar os processos de *Release Planning* na ALERT e qual é o papel do Jira *Software* nesses processos;
- Analisar e estabelecer os requisitos inerentes ao problema proposto pela empresa com o intuito de corretamente definir uma solução para o mesmo;
- Avaliar qual o valor associado ao desenvolvimento do projeto;
- Elaborar diferentes desenhos da solução a implementar. Discutir e definir desenho final.
- Determinar quais as técnicas de avaliação a aplicar ao projeto desenvolvido.

Na segunda fase, orientada ao desenvolvimento do projeto tendo em conta o trabalho realizado na primeira fase, foram definidas as seguintes tarefas:

- Elaborar o desenho final da solução;
- Desenvolver os artefactos para os requisitos definidos na primeira fase e realizar a sua implementação;

- Experimentar, testar e avaliar os requisitos implementados;
- Analisar a solução e aferir se os objetivos foram atingidos.

1.7.1 Sistema de Controlo de Versões

Os sistemas de controlo de versão são ferramentas de *software* que auxiliam a gestão das alterações no código-fonte com o decorrer do tempo [8]. Para este projeto foram considerados dos sistemas de controlo de versões, *Apache Subversion (SVN)* [9] para o código-fonte do *backend* e Git [10] para o código-fonte do *frontend*.

1.7.2 Design Science Research

Design Science Research é uma metodologia de pesquisa baseada em resultados, que oferece linhas específicas para avaliação e iteração em projetos de pesquisa [11].

Deste modo, apesar de a metodologia de trabalho ter ido ao encontro do que é praticado na ALERT, no que diz respeito à metodologia cientifica não houve restrições. Assim, *Design Science Research* surge como a abordagem que melhor se enquadra no contexto desta dissertação.

Ainda que, *Design Science Research* seja uma metodologia só, há diferentes propostas para se conduzir pesquisas com base nela. Contudo, a sua generalidade, tem um único objetivo: desenvolver um artefato e conhecimento técnico-científico.

Posto isto, para o desenvolvimento deste projeto optou-se por seguir a *Information Systems Research Framework* [12], por se considerar ser a melhor abordagem dado contexto.

1.8 Estrutura do documento

O documento está dividido em sete capítulos. Para além do presente Capítulo, existe o Capítulo 2, que está divido em 3 partes principais, as duas primeiras partes apresentam o contexto e o estado da arte de duas temáticas associadas a esta dissertação. Na última parte, é apresentada uma análise às ferramentas existentes no mercado relativas a este tema. No Capítulo 3, é realizada uma análise de valor da solução pensada, sendo o principal objetivo apresentar qual é o valor real que a solução gera para quem irá usufruir desta plataforma.

O Capitulo 4, diz respeito à análise do problema onde serão apresentados os detalhes do mesmo, nomeadamente as partes interessadas na plataforma, os atores da plataforma, as integrações necessárias, quais os requisitos necessários para solucionar o problema e o domínio do problema. Posto isto, será apresentado, no Capítulo 5 a arquitetura da solução que visa dar resposta aos objetivos descritos no Capitulo 1. Nesta capitulo, são apresentados alguns artefactos de engenharia de *Software* que irão permitir ao leitor compreender melhor a arquitectura da plataforma desenvolvida, também aqui serão apresentadas algumas alternativas para a arquitetura. serserá descrito o processo de implementação do *software* que visa responder ao requisitos definidos anteriormente. Sendo que, a construção desta arquitetura é descrita no Capítulo 6

Por último, serão apresentadas as técnicas adotadas para avaliação da solução e os resultados da mesma serão descritas no Capítulo 7 e no Capitulo 8, o capitulo que fecha este documento, será exposta uma reflexão acerca de todo o trabalho desenvolvido.

Capítulo 2

Contexto e Estado da Arte

O presente capítulo pretende contextualizar o leitor sobre os conceitos de *Software Release Planning* e *Schedule Network Analysis*. Para além disto, será apresentado um conjunto de ferramentas existentes relacionadas com o processo de *Software Release Planning*.

2.1 Software Release Planning

Esta secção pretende dar uma contextualização do conceito de *Software Release Planning*.

Inicialmente, será realizada uma introdução teórica acerca do tópico *Software Release Planning*, abordando de uma forma sintetizada em que consiste um *Software Release Planning*, qual o seu propósito, a sua importância e quais os conceitos e as dificuldades associadas a si.

Seguidamente, será apresentado o processo de *Release Planning* no contexto da ALERT, onde será explicada qual a abordagem que a ALERT adoptou no seu processo de *Release Planning*.

Por fim, surge uma apresentação de trabalhos relacionados que propõe abordagens diferentes para o processo de *Software Release Planning*.

2.1.1 Introdução Teórica

Release Planning é um processo que surge associado ao planeamento de um produto incremental de *software* e incluí a atribuição de requisitos à *release* da forma mais benéfica dentro do esforço, orçamento e tempo disponíveis [13].

Assim sendo, *Release Planning* reúne um vasto conjunto de decisões como, por exemplo, *Quais os requisitos que irão ser contemplados na versão?*, *Como maximizar os recursos disponíveis para uma determinada versão?*, *Como distribuir os requisitos pelos recursos disponíveis?*, etc... [2]

Pelo que, o principal objectivo de um *Release Planning* é garantir que a versão de *software* em planeamento oferece o melhor valor comercial para o investimento que foi realizado, o que implica uma selecção acertada dos requerimentos [14].

Resumindo, *Software Release Planning* é uma das partes mais importantes para o desenvolvimento de um produto incremental de *software* e para ser considerado bom deve [2]:

- Fornecer o máximo valor de negócio, permitindo que seja disponibilizada a melhor combinação possível de funcionalidades;

- Satisfazer as partes interessadas;
- Ser viável com os recursos disponíveis;
- Refletir as dependências entre funcionalidades.

Planeamento Estratégico e Planeamento Operacional

O processo de *Release Plannig* pode ser divido em dois: planeamento estratégico e planeamento operacional.

O planeamento estratégico corresponde à seleção e triagem de quais os requisitos que irão ser contemplados numa determinada *release*. Esses requisitos podem surgir de várias fontes, sendo as mais comuns o cliente, os utilizadores e a equipa de suporte técnico que conhece o produto de software e sabe os *bugs* e as necessidades às quais o *software* ainda não dá resposta. Através de negociações entre as partes interessadas e de uma análise aos requisitos resulta uma lista de quais os que irão ser incluídos na *release*, e com que prioridade. No final deste processo é espectável que exista uma *scope* da *release*, isto é, o conjunto de requerimentos a ser implementados [3].

Concluído o planeamento estratégico que resultou numa lista de requerimentos segue-se o planeamento operacional. O planeamento operacional corresponde à decisão de quais as equipas de recursos humanos envolvidas no desenvolvimento da *release* e à distribuição das funcionalidades pelas mesmas [3].

Ou seja, planeamento estratégico responde à questão *O que fazer?* e o planeamento operacional responde às questões *Quem vai fazer?* e *Quando se vai fazer?*.

Dificuldades do Processo de Software Release Planning

O processo de *Release Planning* é um processo complexo pois, selecionar e escalonar funcionalidades de forma otimizada é complexo. Além disso, a execução de um planeamento é algo dinâmico que depende de vários fatores e obriga, constantemente, a um replaneamento [2].

A priorização dos requerimentos, a dependência entre requisitos e as constantes alterações aos requisitos são alguns dos aspetos que tornam o processo de *Release Planning* complexo e desafiante [14].

Porém, muitas outras tarefas exigentes tornam este um processo muito complexo, como estimativa de recursos, definição de objetivos, a estimativa do esforço necessário aplicar num requerimento, a coordenação de equipas, alocação de recursos, etc... [15]

A Figura 2.1 apresenta de forma categorizada e sintética quais são os desafios mais comuns associados ao processo de *Software Release Planning*.

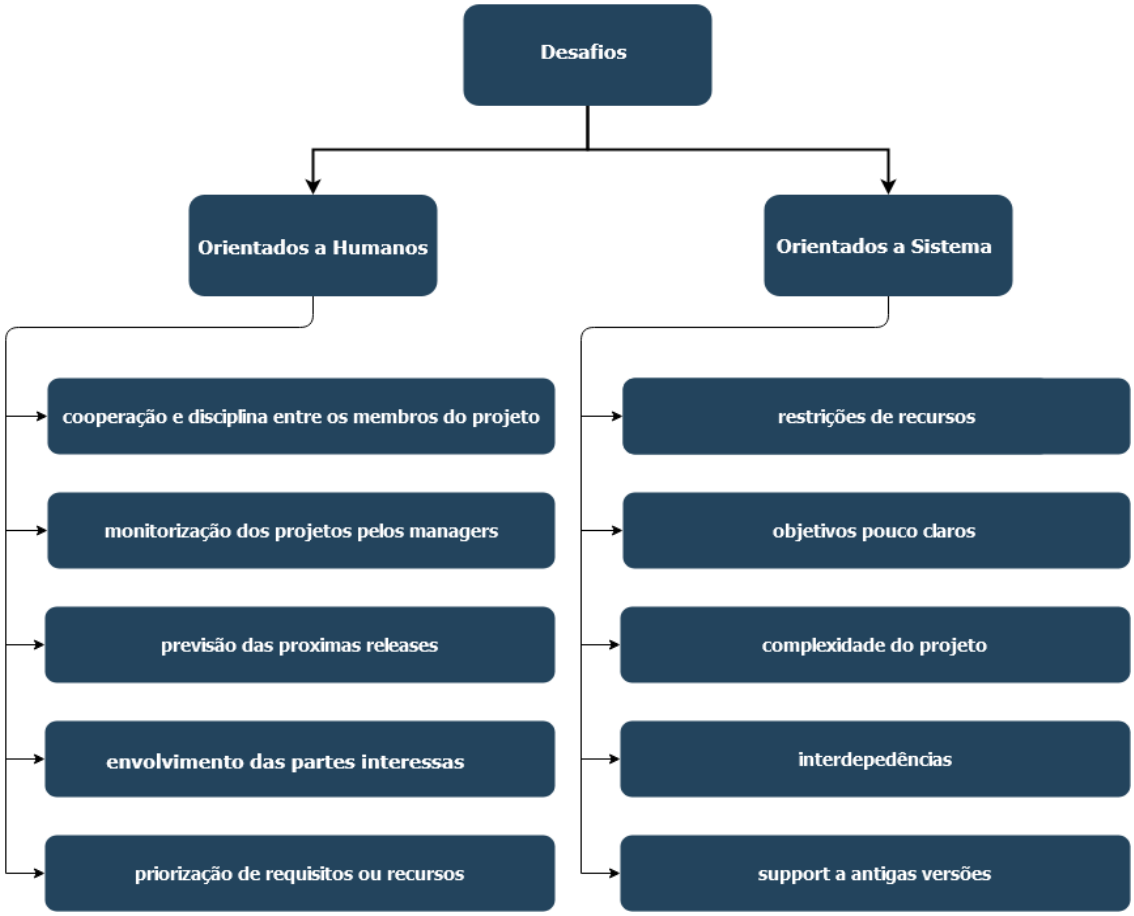

Figura 2.1: Sintetização dos desafios do processo de *Software Release Planning* [adaptado de [15]]

2.1.2 Processo de Software Release Planning na ALERT

Esta secção tem como objectivo apresentar o processo de *Software Release Planning* implementado na ALERT. De modo a que seja possível apresentar com clareza este processo é necessário apresentar também a estrutura da equipa de desenvolvimento da ALERT bem como o processo de desenvolvimento de produto da ALERT, onde está inserido o processo de *Release Planning*.

Estrutura da equipa de desenvolvimento

A equipa de desenvolvimento da ALERT está dividida em quatro equipas - *Functional Analysis & Design*, *Code Development*, *Content* e *Quality Control* - que, por sua vez, podem estar divididas em mais equipas.

A Figura 2.2 demonstra como está organizada a equipa de desenvolvimento da ALERT.

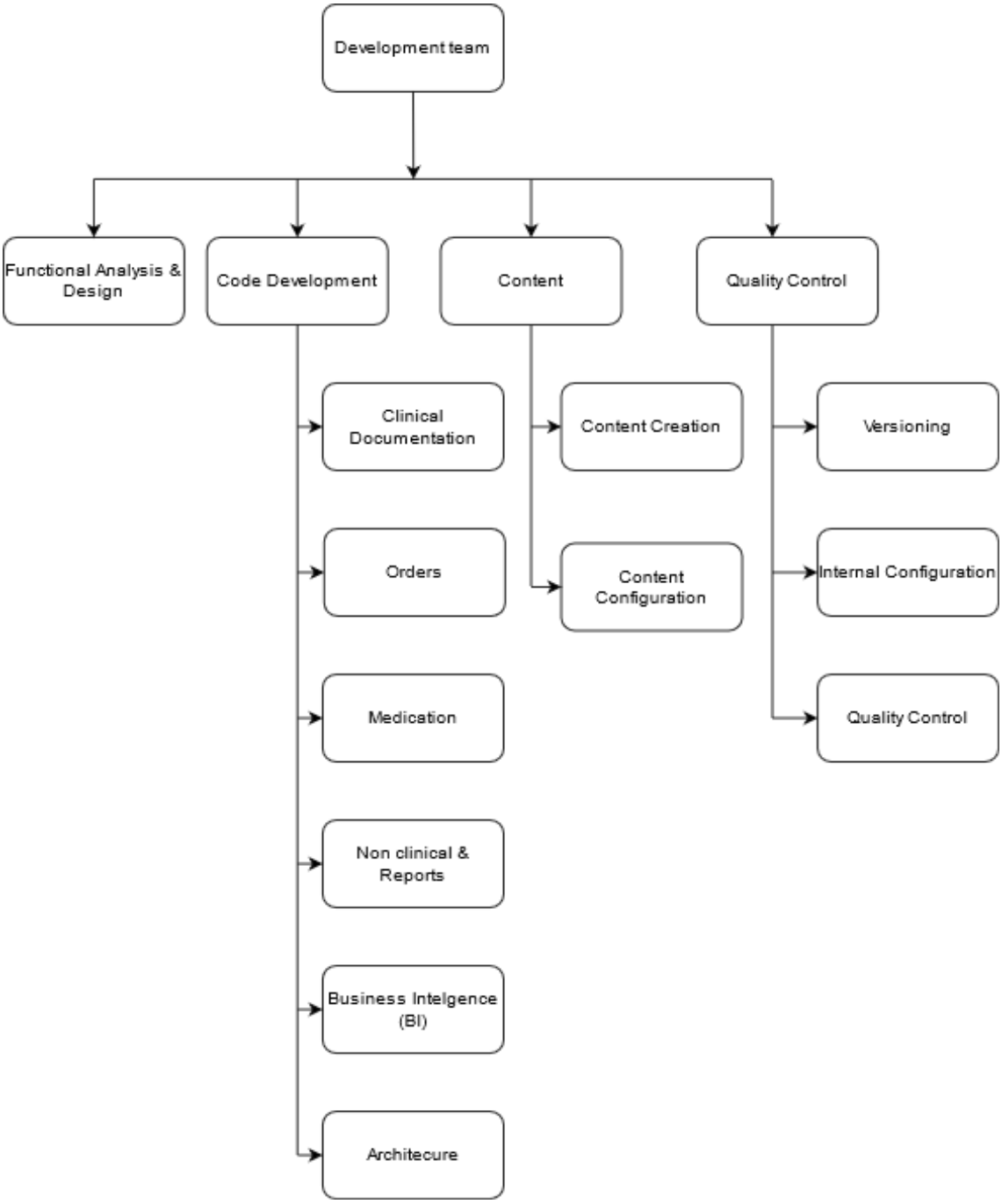

Figura 2.2: Equipa de Desenvolvimento da ALERT

A *stack* tecnológica da ALERT (Javascript, Angular, HTML5, Java, PL/SQL e Base de Dados Oracle) levou a cada equipa de *code development* esteja dividida por camadas tecnológicas, sendo estas *User Interface/User experience (UI/UX)*, *Middleware (MW)* e *Data Base (DB)*.

Resumindo, a equipa e de desenvolvimento da ALERT é uma equipa divida em outras pequenas equipas com papeis bem definidos no desenvolvimento do seu produto.

Desenvolvimento do ALERT®

A ALERT é, como já mencionado, uma *software house* que desenvolve diretamente todas as fases do ciclo de vida da sua *suite* de produtos. Cada desenvolvimento das diferentes linhas de produto ALERT® é orientado a uma única linha de versões que segue um processo maturado que compreende: Recolha de Requisitos (internos ou externos), Planeamento, Desenvolvimento Controlo de Qualidade e Versionamento.

O *workflow* de desenvolvimento do ALERT® tem a Certificação ISO 9001 [16] e está ilustrado na Figura 2.3.

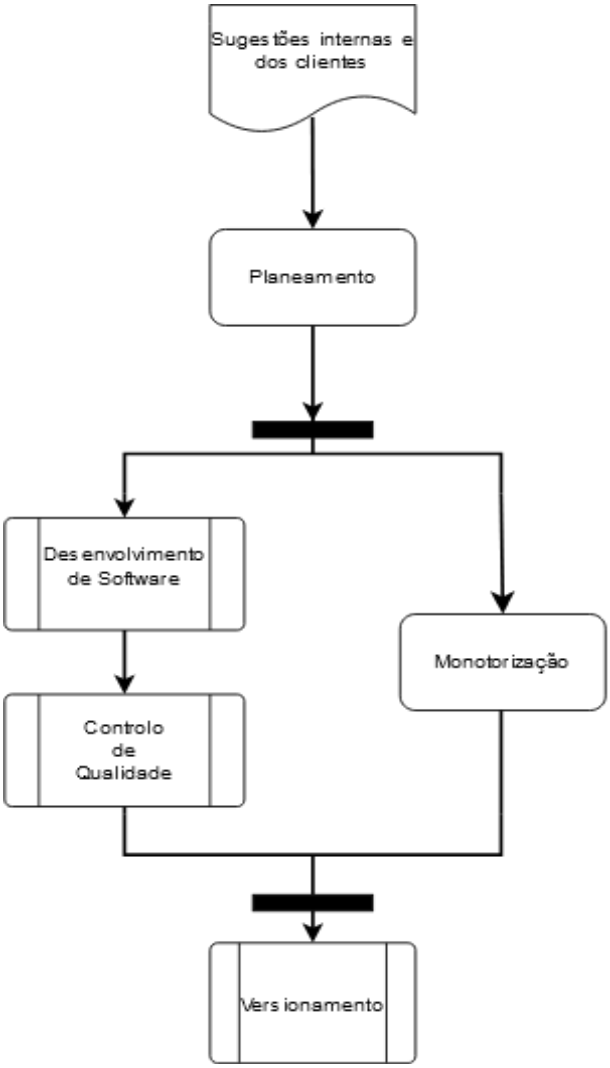

Figura 2.3: *Workflow* do Desenvolvimento

Fase de Release Planning

Há diversas abordagens para o problema de *Release Planning*. A ALERT tem o seu processo de *Release Planning* e que irá ser apresentado de seguida.

Após a recolha de requisitos, todos são analisados e, se aprovados, dão origem a um *issue*[1] no Jira[2].

Cada processo de *Release Planning* inicia-se com a recolha de informação de prioridades operacionais. Nesta fase, um conjunto de *issues* anteriormente especificados são planeados e associados à *release* que irá iniciar o seu desenvolvimento, ou seja, é definido o *scope* da *release*.

Definido o *scope*, todos os responsáveis das diferentes equipas de desenvolvimento estimam o esforço necessário de cada *issue* por equipa e por camada (UI/UX, MW, DB).

Concluído este passo, todas as estimativas são recolhidas de modo a obter-se o total de esforço estimado mais elevado para um determinada equipa e camada. Este valor é depois dividido pelo número de recursos humanos alocados aquela *release* de modo a obter o número de dias úteis mínimos necessários para o desenvolvimento de todos os *issues* definidos no *scope*. A soma dos dias úteis previamente calculados com a data de inicio do desenvolvimento da *release* resulta numa primeira data prevista para o fim do desenvolvimento.

Dado que, as equipas de desenvolvimento têm um número diferente de recursos alocados à versão, o processo de cálculo do número mínimo de dias úteis necessários é repetido quantas vezes necessárias até achar a data estimada ideal.

Quando obtida a data estimada ideal dá-se por concluída a fase de *Release Planning*, tendo sempre em atenção que todo o processo de desenvolvimento e controlo de qualidade requer constante monitorização e diversos factores obrigam a um *replanning* da *release* o que, por vezes, obriga um constante repetição do processo aqui apresentado durante o ciclo de vida de uma *release*.

A Figura 2.4 é uma fluxograma que apresenta uma vista simplificada do processo de *Release Planning* da ALERT.

[1] No Jira, *issues* normalmente representam novas funcionalidades, requisitos do utilizador e *bugs* de software.

[2] Ferramenta que permite a monitorização de tarefas e acompanhamento de projetos.

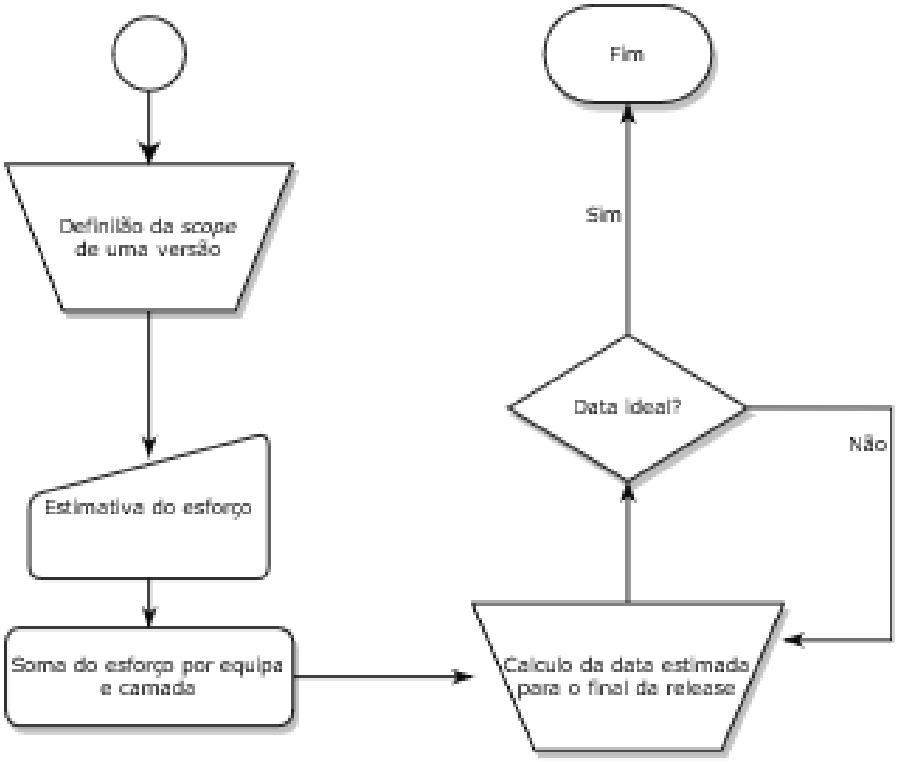

Figura 2.4: Fluxograma do processo de *Release Planning* da ALERT

2.1.3 Outras Abordagens

Para além da abordagem da ALERT apresentada, uma abordagem que depende da intuição, comunicação e capacidade de negociação dos recursos humanos envolvidos no processo, existem outras abordagens e propostas para a o problema do *Release Planning*.

Através de pesquisa relacionada com o estado da arte do domínio deste problema foi possível identificar que há diversas propostas de como o problema deve ser encarado. Vários autores propuseram abordagens e estratégias para resolver os problemas relacionados com a escolha e priorização de requisitos durante o *Software Release Planning* [17] [18] [19], outros foram além deste problema e propuseram resolver não só problemas relacionados com a selecção e priorização de requisitos mas também com o escalonamento de requisitos [20] [21].

Em seguida, serão apresentadas, primeiramente, as abordagens que propõem resolver os problemas de escolha e priorização de requisitos e, depois, as abordagens que propõe combinar a escolha e priorização de requerimentos com o seu escalonamento.

EVOLVE*

Em *"Intelligent Support for Software Release Planning"* [17], Amandeep, Günther Ruhe, e Mark Stanford afirmam que o principal objetivo do *Release Planning* para o desenvolvimento de *software* incremental é encontrar um conjunto de requerimentos para o incremento a ser desenvolvido que maximizasse a soma de todas as prioridades de todas as partes interessadas no incremento.

Neste sentido, propuseram uma solução evolucionária - EVOLVE* - que combina a potencialidade dos algoritmos genéticos com a a flexibilidade de um método iterativo. No EVOLVE*, numa determinada iteração k, uma decisão é feita sobre o incremento imediato inc^k.

O EVOLVE* tem como foco os processos incrementais de desenvolvimento de *software* e as principais vantagens são:

- Levar em consideração as prioridades das partes interessadas, bem como as restrições de esforço para todas as *releases*;
- Fornecer suporte inteligente para *Software Release Planning*;
- Ter em consideração a precedência inerente, o acoplamento e as restrições de requisitos;
- Capacidade de lidar facilmente com alterações nos requisitos e nos atributos do projeto;
- Atender a conflitos nas prioridades das partes interessadas, reconhecendo que as opiniões das partes interessadas nem sempre são iguais;
- Gerar um conjunto de soluções mais promissoras (em vez de apenas uma solução).

Para além da apresentação desta proposta, os autores apresentaram os resultados da aplicação de uma ferramenta inteligente com base neste modelo num ambiente real de desenvolvimento (IGrafx Corel Inc.) para validarem o cenário de melhoria.

EVOLVE^{ext}

O EVOLVE^{ext} é descrito em *"Strategic Release Planning and Evaluation of Operational Feasibility"* [18] como extensão do método anteriormente apresentado (EVOLVE*).

Esta abordagem apresentada por Guenther Ruhe e Joseph Momoh combina inteligência computacional e humana para resolver as dificuldades inerentes do problema *Software Release Planning*:

- Os requisitos não são bem específicados e/ou compreendidos;
- Mudança de requisitos;
- As opiniões entre as partes interessadas competem entre si;
- A complexidade do problema;
- A incerteza dos dados relacionados com o esforços, os recursos e os riscos inerentes.

EVOLVE^{ext} foca-se na interacção entre o planeamento estratégico e operacional e difere devido ao planeamento horizontal de tempo, aos objetos a serem planeados e à granularidade do planeamento. Para além disto, esta extensão também oferece um sistema mais flexível de votação das partes interessadas. Com esta extensão, o modelo permite que as partes interessadas no sistema de *software* possam realizar uma votação mais flexível relativamente aos requisitos que devem integrar uma determinada *release*. O novo modelo abrange grupos de requisitos e permite a atribuição flexível de partes interessadas a requisitos ou grupos de requisitos.

Além do mais, nesta abordagem, a viabilidade operacional de um *Release Plan* proposto é avaliada para estender as capacidades do plano estratégico. Os recursos são uma parte importante das *releases* e é essencial considerar todos os tipos de recursos disponíveis para

uma *release*. Nesta abordagem, são formulados três tipos de problemas de viabilidade - Verificação da viabilidade; Extensão ideal de recursos; Redução ideal de funcionalidades - para validar a viabilidade de o plano estratégico proposto para o próximo lançamento imediato do incremento de *software* com atenção às tarefas e aos recursos disponíveis.

No final do artigo, os autores afirmam que o EVOLVE^{ext} é mais adequando para problemas de *Release Planning* grandes e complexos trazendo mais benefícios para as organizações que possuam processos maturados.

DECIDE$_{\text{Release}}$

Os problemas de decisão em engenharia são incertos por natureza, afirmam Ahmed Al-Emran, Dietmar Pfahl e Guenther Ruhe no seu artigo *"Decision Support for Product Release Planning based on Robustness Analysis"* [19]. Para além disto, defendem que uma solução é considerada robusta se permanecer válida após as alterações nos parâmetros do problema, ou seja, a robustez associado à tomada de decisão refere-se à estabilidade das mesmas perante a presença de incerteza.

Neste sentido, os autores propuseram um método chamado DECIDE$_{\text{Release}}$ que, através dos dados resultantes da avaliação à robusteza dos planos operacionais, toma decisões relativas ao plano estratégico da *release*.

A avaliação à robustez recai na investigação sistemática do impacto da incerteza nas estimativas de esforço e na produtividade. Depois, os dados resultantes da investigação são utilizados para escolher um plano estratégico robusto e à prova de mudanças.

O DECIDE$_{\text{Release}}$ é, então, um novo método que facilita a tomada de decisões estratégicas e facilita a integração entre os planos estratégicos e operacionais. Tendo por base a avaliação de planos estratégicos obtidos através do EVOLVE*, este método fornece suporte qualificado à tomada de decisão por meio de combinação de análise pro-ativa baseada em simulação e aplicação de análise de decisão multi-critério.

Como prova de conceito, a aplicabilidade do método proposto foi demonstrada através de uma caso de estudo e os resultados foram apresentados.

OPTIMIZE$_{\text{RASORP}}$

OPTIMIZE$_{\text{RASORP}}$ é uma abordagem de optimização composta por duas fases que combina dois métodos e visa resolver o problema resultante que advém da alocação de recursos a uma *release*, *"Resource Allocation for Software Release Planning"(RASORP)*. Esta abordagem está descrita no artigo da co-autoria de An Ngo-The e Günther Ruhe *"Optimized Resource Allocation for Software Release Planning"* [20].

An Ngo-The e Günther Ruhe identificam que "O grande problema enfrentado pelas empresas que desenvolvem ou mantêm sistemas grandes e complexos é determinar quais elementos de um conjunto tipicamente grande de recursos candidatos devem ser atribuídos a quais versões do software."e como tal propõe uma solução de duas fases que combina *Integer Linear Programming (ILP)* - Fase 1 - com a potencialidade dos algoritmos genéticos, do inglês *Genetic Algorithms (GAs)* - Fase 2 - que resolve de forma otimizada o problema RASORP.

Formalmente, An Ngo-The e Günther Ruhe definem RASORP como,

$$\text{Maximize } \{F(x) = \sum_{n=1...N} \sum_{k=1...K} v(n,k) \cdot x(n,k) \text{ subject to } (x,u) \in (X,U)\} \quad (2.1)$$

Em que x é um *release plan* associado a uma alocação de um recurso u e forma o vetor combinado (x, u). O conjunto de todas as possibilidades (x,u) é o produto Cartesiano $X \cdot U$. E v(n, k) é a contribuição expectável de uma determinada funcionalidade caso seja associada a uma *release* k. E x(n, k) as funcionalidades atribuídas a uma *release*.

Assumindo que cada funcionalidade tem um valor e um esforço associado a si e que cada recurso necessário à sua implementação tem uma capacidade, os autores procuram, através do seu método, encontrar um subconjunto de funcionalidades que possa ser considerado para uma *release* de modo a que o valor do subconjunto seja o máximo onde o esforço necessário não exceda a capacidade existente.

Assim, na Fase 1, através da aplicação de ILP, procuram resolver o problema 2.1, de modo a obter o conjunto de funcionalidades que maximizem o valor da *release*. Já na Fase 2 o objetivo é, com recurso a GAs, procurar a melhor solução para atribuir recursos humanos a tarefas das funcionalidade escolhidas para a *release*. No final, cada tarefa é atribuída a um recurso humano que consiga acabar a tarefa o mais cedo possível.

Para além da formalização do problema e da proposta de solução, é possível encontrar os resultados da aplicação do modelo num caso de estudo hipotético.

No final, os autores salientam que:

- o problema de *release planning* combinado com o problema da alocação de recursos a tarefas nunca tinha sido abordado anteriormente;
- os niveis de produtividade do recurso humano são um fator importante em casos reais;
- a combinação de GA com ILP reduz o espaço de pesquisa e supera as fragilidades dos GAs;
- a solução obtida apresenta um grau de otimização;
- o seu método, mostrou ser aplicável a problemas com até 200 recursos e 600 tarefas.

An integrated approach for requirement selection and scheduling in software release planning

Chen Li, Marjan van den Akker, Sjaak Brinkkemper e Guido Diepen autores de "*An integrated approach for requirement selectionand scheduling in software release planning*"[21] afirmam que há uma extensa pesquisa relativamente ao problema da seleção de requerimentos, comparativamente com o escalonamento de dos requerimentos. Pelo que, com este artigo, visam apresentar dois modelos ILP que integram seleção de requerimentos com escalonamento.

O primeiro modelo que apresentam, oferece uma técnica de escalonamento que fornece um conjunto de requerimentos com vista a minimizar a duração da *release*. O segundo modelo combina a seleção de requerimentos com o escalonamento, onde a maximização do valor da *release* é combinada com um escalonamento do projeto dentro do prazo da *release*.

Os autores desenvolveram também um protótipo que apresenta uma implementação de um modelo de seleção de requerimentos, de um modelo de escalonamento e da combinação dos dois modelos.

Para o problema do escalonamento, foi aplicado um modelo *Resource Constrained Project Scheduling Problem (RCPSP)* e para o problema da escolha de requerimentos foi aplicado um modelo *Knapsack*.

São, também apresentados, dois mecanismos que permitem ajustar dinamicamente a *release* após o início da mesma. O primeiro mecanismo permite modificações em tempo de execução do modelo e o segundo aplica o conceito *scrum* numa metodologia ágil de desenvolvimento.

Para além disto, os autores concluem que a selecção de requisitos combinado com o escalonamento não consegue gerar apenas uma única solução, porém, ainda assim, é mais eficiente que o tradicional método *Knapsack*.

Sumário

Independentemente da abordagem que se adopte, *Release Planning* não pode ser resolvido meramente com recurso às potencialidades de computação. Günther Ruhe e Moshood Omolade propõe com o artigo *The Art and Science of Software Release Planning* [2], que se crie uma sinergia entre a arte e ciência para a resolução do problema de *Release Planning*, sendo que, a inteligência computacional deve ser aplicada na formalização do problema e a inteligência humana aplicada na resolução problemas táticos do *Release Planning*.

De uma forma sintetizada, o que eles sugerem é que as potencialidades computacionais devem ser utilizadas para gerar um conjunto de soluções para uma sequência de problemas, em substituição da determinação de apenas um plano para um problema vagamente definido. Gerado o conjunto de soluções alternativas "suficientemente boas", este é apresentado a um humano que avalia as alternativas com base na sua experiência e familiaridade com o contexto do problema.

Em suma, os autores defendem que uma abordagem deste tipo elevaria o nível de como são realizados os *Release Plans*.

2.2 Schedule Network Analysis

Schedule Network Analysis é uma técnica de análise que gera um cronograma[3] para um determinado projeto. Utiliza várias técnicas de análise como por exemplo, *Critical Path Method (CPM)*, *Program Evaluation and Review Technique (PERT)* e *Critical Chain Method (CCM)* para calcular as datas antecipadas de inicio e de fim para as diferentes atividades que compõe um projeto [7]. De seguida, serão introduzidas as técnicas que podem ser relevantes para o contexto do problema desta dissertação.

2.2.1 Critical Path Method

Critical Path Method é o atual método de análise utilizado na ALERT para estimar a data final do desenvolvimento de uma *release* e obter o conjunto de atividades críticas e

[3]Representação gráfica do calendário de um plano ou projecto. "cronograma", in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2021, https://dicionario.priberam.org/cronograma [consultado em 12-02-2021].

corresponde a um conjunto de técnicas de modelação e escalonamento que ajudam a planear, controlar e prever o rumo de um determinado projeto. Tipicamente, a aplicação do CPM divide um projeto complexo em ativadades, cuja a duração e sequência podem ser definidas. [22].

Como já referido, o CPM é utilizado para estimar a duração mínima de um projeto. O termo caminho crítico refere-se ao conjunto de atividades que representam o caminho mais longo num projeto, ou seja, a duração mais curta possível do projeto [7].

A aplicação deste método deve ter em consideração [23]:

1. quais as tarefas a ser realizadas;
2. a sequência pela qual serão realizadas;
3. os recursos necessários à sua realização;
4. e o tempo necessário para cada tarefa.

2.2.2 Program Evaluation and Review Technique

Program Evaluation and Review Technique é uma técnica de análise de projetos que considera o projeto a analisar, assim como no método CPM, como uma rede acíclica de eventos e atividades [24].

Um dos aspestos que distingue o PERT do CPM são as das durações das atividades, enquanto que o CPM tem uma única estimativa para a duração, no método PERT a duração tem um valor esperado e um desvio padrão.

Assim, após a identificação das tarefas do projeto e estabelecidas as suas relações, são definidas três estimativas para cada tarefa: otimista (o), pessimita (p) e a mais provável (m). Onde a estimativa otimista é o cenário perfeito, onde tudo corre bem; a estimativa pessimista é o pior cenário, onde tudo corre mal; a estimativa mais provável é onde tudo corre dentro da normalidade, sem percalços. [25]

Para calcular a duração espectável (μ) recorre-se à equação PERT [25]:

$$\mu = (a + 4m + b)/6 \tag{2.2}$$

e para o cálculo do desvio padrão da duração de uma tarefa (σ^2) recorre-se à equação [25]:

$$\sigma = (b - a)/6 \tag{2.3}$$

A duração mínima do projeto é obtida através da identificação das actividades que estão no caminho crítico, como no CPM. Aliás, as semelhanças entre o PERT e o CPM são de tal ordem que várias vezes PERT é referenciado como PERT/CPM [26].

2.2.3 Critical Chain Method

O Critical Chain Method é em tudo semelhante ao CPM, a única diferença é que este método permite a adição de *buffers* de duração em qualquer um dos caminhos do projeto para que se possa contabilizar os recursos limitados e as incertezas do projeto [7].

Buffers de duração são tarefas não relacionadas com o trabalho que são adicionadas à *network* do projeto de maneira a lidar com a incerteza [7]. Os *buffers* são extraídos da duração estimada das actividades. Os *buffers* podem ser *buffer* do projeto, que agregam a "segurança"no final do projecto e atuam como protecção da data final do projeto ou *buffers* de alimentação, que são *buffers* posicionados onde as tarefas não críticas se juntam à cadeia crítica, de modo a proteger as tarefas críticas contra as variações negativas das cadeias que a alimentam [27].

Existem diversos métodos para estabelecer o tamanhos dos *buffers*, e o artigo *"Critical chain buffer sizing: a comparative study"* [27] apresenta uma perspetiva comparativa entre eles.

Este método adota uma abordagem do método do caminho crítico porém, tem em consideração factores como alocação de recursos, otimização de recursos, nivelamento de recursos e incerteza da duração das atividades que compõe o caminho crítico determinado [7].

Assim, pelo facto de alguns recursos terem disponibilidade limitada, o caminho crítico resultante provavelmente será mais longo do que o caminho crítico obtido com o algoritmo do CPM, surgindo o conceito de cadeia crítica [28]. Cadeia crítica é, então, o caminho crítico com restrição de recursos.

2.2.4 Sumário

Em suma, nesta secção são apresentadas três técnicas de análise onde um projeto é visto como uma *network* de atividades. Estas técnicas adotam procedimentos distintos para atingir um objetivo comum: obter o conjunto de atividades críticas afetas a um projeto de modo a estimar o final do mesmo.

Independentemente do método utilizado, todos fornecem a possibilidade de compreender melhor quais as atividades/tarefas em que é necessário mais foco, para que seja possível reduzir significativamente a hipótese de uma projecto atrasar.

No final, o estudo realizado e aqui documentado permitiu ao Aluno compreender os métodos de análise de projetos. Esta compreensão irá permitir que, aquando o desenvolvimento da solução, seja possível adotar a melhor metodologia de modo a que, dado o contexto do problema, seja possível desenvolver a solução mais capaz de cumprir com o objetivo de calcular uma data prevista para o fim do desenvolvimento das *release* do ALERT ®EMR.

2.3 Ferramentas de apoio ao Software Release Planning

Nesta secção irão ser apresentadas algumas ferramentas que foram desenhadas e construídas de modo a dar suporte ao processo de *Software Release Planning*. A escolha das ferramentas teve em conta os seguintes aspetos: ter uma versão grátis, ter estimativas de tempo e ter gestão de recursos.

2.3.1 Axosoft

Axosoft [29] é uma ferramenta de gestão de projetos e rastreamento de *bugs*. É uma plataforma de gestão ágil de projetos bastante utilizada por desenvolvedores que utilizam metodologias *Scrum*.

Esta plataforma oferece um conjunto de ferramentas que visam garantir a criação e entrega de *software* totalmente funcional e sem erros dentro dos prazos estabelecidos. Destas ferramentas destaca-se o *Release Planner*.

Release Planner oferece uma nova abordagem para a alocação de recursos em *sprints* e *releases*. O objetivo é minimizar o tempo que é adicionado a uma *release* por mau planeamento. Esta funcionalidade oferece a possibilidade de visualizar quem está disponível e quanto trabalho pode ser planeado para um determinado *sprint* ou uma determinada *release*.

2.3.2 TeamGantt

TeamGantt [30] é uma ferramenta de gestão de projetos através de gráficos de Gantt[4]. Esta plataforma é utilizada por empresas como Amazon, Netflix, Nike e Disney, e a sua popularidade resulta da facilidade de utilização que oferece.

Destacam-se funcionalidades como: possibilidade de visualizar a disponibilidade dos recursos, acompanhamento do progresso do projeto em *real-time*, histórico dos projetos, definição de dependências entre tarefas de um projeto, comparação entre as projeções do cronograma original com o cronograma real do projeto, entre outras...

2.3.3 TeamWork

TeamWork [31] é uma plataforma online que facilita a gestão de projetos e todas as suas tarefas, fornecendo a todos os membros das equipas uma interface clara e com as ferramentas que necessitam.

Oferece quatro tipos de subscrição:

- Free Forever;
- Pro;
- Premium;
- Enterprise.

Entre as funcionalidades que oferece, destaque para os gráficos de Gantt interativos que permitem visualizar todas as tarefas do projeto, as suas dependências, os seus prazos e para a funcionalidade *Workload* que numa ferramenta de gestão de recursos que permite visualizar a quantidade de trabalho realizado e as disponibilidades dos recursos.

2.3.4 VersionOne

VersionOne [32] é uma ferramenta de gestão de projectos ágeis que se adapta rápido a qualquer metodologia ágil de desenvolvimento de software.

Esta ferramenta, oferece uma plataforma sólida de planeamento para apoiar o desenvolvimento ágil e incorpora a garantia de qualidade como uma parte essencial de todo o processo de desenvolvimento de *software*.

VersionOne tem várias edições que apresentam funcionalidades diferentes:

- Team;

[4] Gráfico de barras usado para rastrear as dependências e progresso das atividades de um projeto.

- Catalyst;
- Enterprise;
- Ultimate.

As funcionalidades centrais de gestão de projetos são: Planeamento de Produto; *Release Planning*; *Sprint/Interaion Planning*; *Sprint/Iteration Tracking*; *Velocity Trend*.

Release Planning consiste no planeamento e monitorização de atividades. E oferece *Release Scheduling*, *Regression Planning* e *Team Scheduling*.

2.3.5 Wrike

Wrike [33] é uma aplicação poderosa e flexível de colaboração que oferece ferramentas para gestão de projetos. O Wrike começou por ser uma plataforma online para a colaboração entre membros de equipa, gestão de trabalho e gestão de projetos. Atualmente, continua a expandir essas áreas adicionando sempre funcionalidades novas relacionadas.

Oferece os seguintes tipos de subscrição:

- Free;
- Professional;
- Business;
- Enterprise;
- Wrike for Marketers;
- Wrike for Marketers Performance;
- Wrike for Professional Services;
- Wrike for Professional Services Performance.

Destaca-se o leque de possibilidades que oferece para visualizar os projetos: lista, quadro, tabela e gráficos Gantt.

Oferece o Wrike Resource, um *add-on* que permite a gestão de recursos. Este *add-on* incluí a possibilidade de ver quem na equipa está sobrecarregado e quem tem mais disponibilidade para assumir mais tarefas. Oferece, também, a possibilidade de gerar diversos relatórios relativos ao trabalho realizado.

2.3.6 Sumário

Nesta secção foram apresentadas diversas ferramentas que oferecem funcionalidades de gestão de projetos, com destaque para as funcionalidades relacionadas com a gestão de recursos (visualização de disponibilidades, alocações de recursos, etc...) e escalonamento de tarefas.

Esta análise permitiu obter uma perspetiva de quais são as ofertas no mercado para as problemáticas associadas ao processo de *Release Planning*.

Capítulo 3

Análise de Valor

A rápida evolução tecnológica, as crescentes necessidades dos clientes e a competição global são alguns dos fatores que exigem que os novos produtos surjam continuamente a uma taxa cada vez mais acelerada. Com vista a colmatar a pressão exercida no desenvolvimento de novos produtos, esse desenvolvimento deve ser transformado num processo de desenvolvimento contínuo e iterativos com ênfase no valor para o cliente [34].

Análise de Valor é, então, uma abordagem criativa e organizada com o objetivo a identificação de custos que não fornecem qualidade, nem uso, nem funcionalidades para um determinado cliente [35].

Posto isto, este capitulo tem como objetivo apresentar ao leitor a análise o conjunto de modelos e técnicas aplicados ao projeto a desenvolver com vista a aumentar o seu valor com o menor custo, sem nunca por em causa a qualidade.

3.1 Processo de Inovação

O processo de inovação é um processo que compreende três fases [36]: *Fuzzy Front End (FFE)*, *New Product Development (NPD)* e Comercialização, e consiste na geração de uma nova ideia e da sua implementação produzindo um novo produto, processo ou serviço que resulta na geração de lucro para uma organização e um crescimento económico dinâmico da nação [37].

3.1.1 Fuzzy Front End

O FFE é uma fase crítica do processo de inovação pois as escolhas efetuadas durante esta fase, em última instância, podem determinar quais opções de inovação podem ser consideradas para o desenvolvimento e comercialização [38].

FFE consiste, então, num processo linear de três etapas [38]:

1. Execução de um pré-trabalho para descobrir novas oportunidades;
2. Definição do *scope*, onde são realizadas rápidas avaliações económicas relativas ao *marketing* e ao mérito técnico do projeto;
3. Construção de uma caso de negócio detalhado.

3.1.2 New Concept Development

New Concept Developmen (NCD) é um modelo que foi desenvolvido por Koen e visa proporcionar uma melhor clareza ao processo de inovação [39]. Este modelo (Figura 3.1) é composto por três partes principais [36]:

- **Motor:** representa a liderança, cultura e estratégias de negócio de uma organização.
- **Cinco elementos:** identificação de oportunidade, análise da oportunidade geração de ideias, seleção de ideias e definição do conceito.
- **Fatores de influência:** capacidade organizacional e fatores internos e externos que possam estar envolvidos e são incotrláveis pela organização.

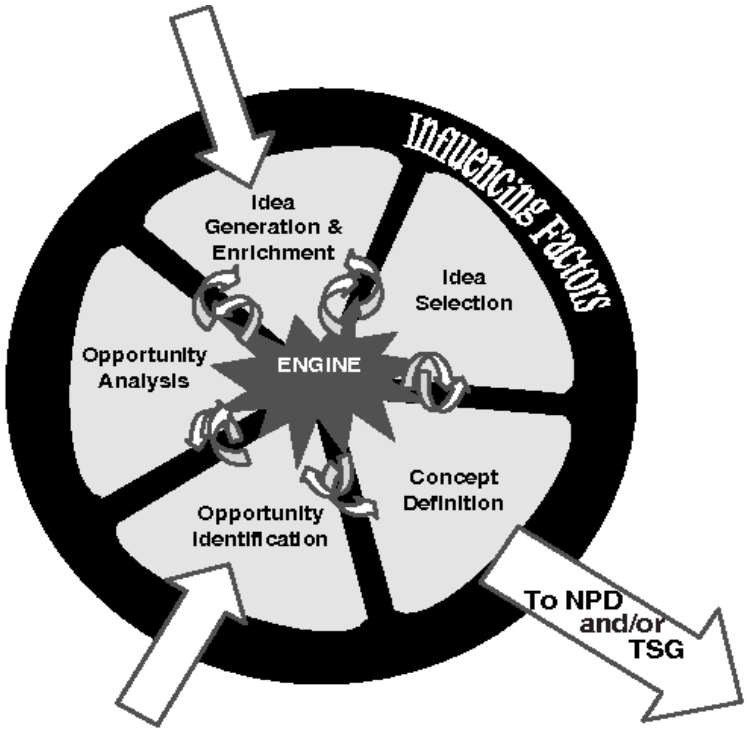

Figura 3.1: Modelo NCD [Fonte: [36]]

Este modelo foi pensado e desenhado como um modelo circular de forma a sugerir que é expectável que as ideias fluam, circulem e iterem entre todos os cinco elementos [39].

Identificação da Oportunidade

Identificação de oportunidade corresponde à primeira fase do Modelo NCD e é, geralmente, motivada pelos objetivos da organização e do negócio. O surgimento de novas oportunidades podem derivar quer de fatores internos como externos.

No contexto desta dissertação, a oportunidade surgiu mediante a necessidade que a equipa de *Release Planning* da ALERT sentiu em obter uma ferramenta capaz de apoiar o processo de *Release Planning* pelo qual estão responsáveis.

Assim sendo, os problemas identificados no Capitulo 1 na Secção 1.2 ilustram a oportunidade que surgiu para agilizar o processo de *Release Planning* da ALERT, através da construção de uma plataforma para esse fim.

Finda a identificação da oportunidade, segue-se uma análise ao seu valor para confirmar que é vantajoso perseguir a oportunidade.

Análise da Oportunidade

Com vista a avaliar a viabilidade da oportunidade identificada anteriormente, procedeu-se à sua análise. Assim sendo, uma análise da oportunidade visa verficar se vale a pena ter em conta uma oportunidade para o cliente [36].

A oportunidade previamente identificada tem o potencial de criar uma plataforma que permita à equipa de *Release Planning* melhorar o seu processo de planeamento sem impactar com o processo em si. Ou seja, há o potencial de ajudar a equipa de *Release Planning* a obter indicadores relevantes para o seu planeamento (em particular, uma estimativa para o final do desenvolvimento) e gerir a alocação dos seus recursos sem prejuízo para o seu atual processo, isto é, sem alterar esse processo. Deste modo, considerou-se que valia a pena persiguir a oportunidade, pois, obtendo sucesso irá facilitar e agilizar o processo de *Release Planning* da ALERT.

Geração de Ideias

A geração de ideias é relativo ao nascimento, desenvolvimento e maturação de uma ideia concreta. As ideias são construídas, destruídas, combinadas, remodeladas, modificadas e atualizadas. Pelo que, a geração de ideias é um processo evolucionário. À medida que uma ideia é estudada, examinada, discutida e desenvolvida esta pode passar por diversas iterações e mudanças [36].

Durante a apresentação da oportunidade e uma discussão com a equipa de *Release Planning* surgiram duas ideias distintas:

- **Ideia 1:** Implementação de uma plataforma que receba os dados das férias e ausências dos recursos humanos de um ficheiro Excel;
- **Ideia 2:** Implementação de um plataforma que fornece-se um módulo de gestão de férias e ausências (marcação de férias, aprovação de férias, etc...).

Seleção da Ideia

Apesar de, ser relativamente fácil gerar ideias, o mesmo não se aplica à escolha da melhor ideia que irá gerar o maior valor garantido o menor custo. Para o projeto em consideração foi aplicado o método *Analytic Hierarchy Process (AHP)* para facilitar o processo de seleção da ideia. A aplicação do método teve por base quatro fatores/critérios: Tempo e esforço de desenvolvimento, Integração com o Jira *Software*, Bom desempenho e Usabilidade/Conveniência. A aplicação do método AHP pode ser consultada no Anexo A. Sendo que, o resultado da sua aplicação recomenda a escolha da Ideia 1.

Definição do Conceito

Este é o último elemento do modelo NCD e serve de porta de saída para os processos de desenvolvimento como o New Product and Process Development (NPPD) [39]. Para este

projeto, esta fase consistiu no desenvolvimento da formalização da presente dissertação. Este documento expõe o problema e os objetivos do projeto, mencionando também os diversos requisitos e restrições.

3.2 Valor para o cliente e percepcionado

3.2.1 Valor para o cliente

O valor para o cliente é frequentemente definido como uma mera relações entre benefícios que um produto visa oferecer com os sacrifícios a si associados [40]. Assim sendo, a Tabela 3.1 apresenta uma comparação entre os Benefícios e Sacrifícios associados a este projeto.

Tabela 3.1: Benefícios vs Sacrifícios

Benefícios	**Sacrifícios**
Redução de erros humanos	Custos associados ao desenvolvimento
Redução do trabalho manual	Custos associados à manutenção da plataforma
Agilização do processo de *Release Planning*	Custos associados à alocação de recursos de infraestrutura para a execução da plataforma
Centralização dos dados das *releases* numa plataforma só de fácil acesso	

3.2.2 Valor para o percepcionado

O valor percepcionado é considerado o valor que o cliente atribui ao produto em questão, tendo em conta o seu negócio [41]. Para o caso de estudo em consideração, o valor percepcionado difere entre as equipas da ALERT. Ao passo que, a equipa *Release Planning* considera uma mais valia para o seu trabalho aumentando a produtividade, agilizando o processo e contribuindo para melhores planeamentos outras equipas, apesar de também puderem tirar partido da plataforma podem não considerar que a plataforma em questão seja assim tão valiosa, pois não representa para eles tantas valias como para a equipa de *Release Planning*. Contudo, esta plataforma representará algum valor para todas as equipas da ALERT pois centraliza uma vasto conjunto de indicadores que permitem acompanhar os progressos de desenvolvimento das *Releases*.

3.3 Proposta de Valor

Proposta de Valor pode ser descrita como uma forma de como um produto ou serviço é preparado e disponibilizado para satisfazer as necessidades de um consumidor [42].

No contexto desta dissertação, a proposta de valor consiste na disponibilização de uma plataforma que visa apoiar o processo de *Software Release Planning* da ALERT com destaque para integração com o Jira *Software* e o calculo de uma estimativa para o final do desenvolvimento de uma *release*.

3.4 Sumário

Este Capítulo teve como intuito a descrição do processo de inovação que contribuiu para formulação deste projeto. Para além disso, foi também identificado o valor que este projeto representa para a ALERT, nomeadamente para as equipas de desenvolvimento e de *Release Planning* da ALERT, de que forma esse valor irá ser percepcionado pelas partes interessadas. No final, é apresentada a proposta de valor que este projeto visa oferecer ao seus utilizadores.

Capítulo 4

Análise do Problema

Após a contextualização do problema e analisado o seu valor é necessário realizar uma análise mas detalhada ao problema em si.

Assim sendo, neste capitulo será apresentada a análise ao problema em questão.

4.1 Domínio do Problema

Como já referido, este projeto surge da necessidade da ALERT sentiu em ter uma plataforma capaz de apoiar o seu processo de *release planning*. Assim, o problema em questão é a criação dessa plataforma.

Para alcançar uma solução é necessário consideração algumas características do processo de desenvolvimento da ALERT:

- a ALERT tem uma *suite* de produtos, cada um desses produtos tem diversas versões, cada versão tem diversos *release plans*;
- cada *release plan* é composto por um conjunto de tarefas;
- tipicamente, as tarefas tem uma estimativa, em horas, para o seu desenvolvimento;
- a ALERT utiliza uma ferramenta, o Jira Software, para fazer o acompanhamento dessas tarefas;
- cada uma das tarefas é realizada por um, ou vários, recursos humanos, como tal, a finalização de cada tarefa está dependente dos constrangimentos de cada recurso humano que a executa.
- cada equipa da ALERT trabalha em paralelo, isto resulta em que, cada equipa executa tarefas que não dependem de outras tarefas.

Em conclusão, para obter uma solução que responda ao problema é necessário desenvovler uma plataforma que integre os dados de cada *release plan*, que se encontram no Jira Software, com dados de recursos humanos para que, no final, uma data prevista para o fim do desenvolvimento desse *release plan* seja calculada. Além disto, esta plataforma deve estar sincronizada com Jira Software para que consiga calcular os indicadores estatísticos de cada *release plan*.

4.1.1 Modelo de Domínio

O Modelo de Domínio representa um conjunto de objetos conectados entre si, onde cada objeto representa um conceito de negócio significativo [43].

A lógica de negócio pode ser algo muito complexo, pelo que um Modelo de Domínio visa modelar em objetos o negócio para o qual se está a desenhar o *software*.

Num Modelo de Domínio é possível encontrar objetos que representam dados do negócio bem como objetos que capturam regras de negócio [43].

Assim sendo, a Figura 4.1 apresenta um Modelo de Domínio que modela o problema em apreço nesta dissertação.

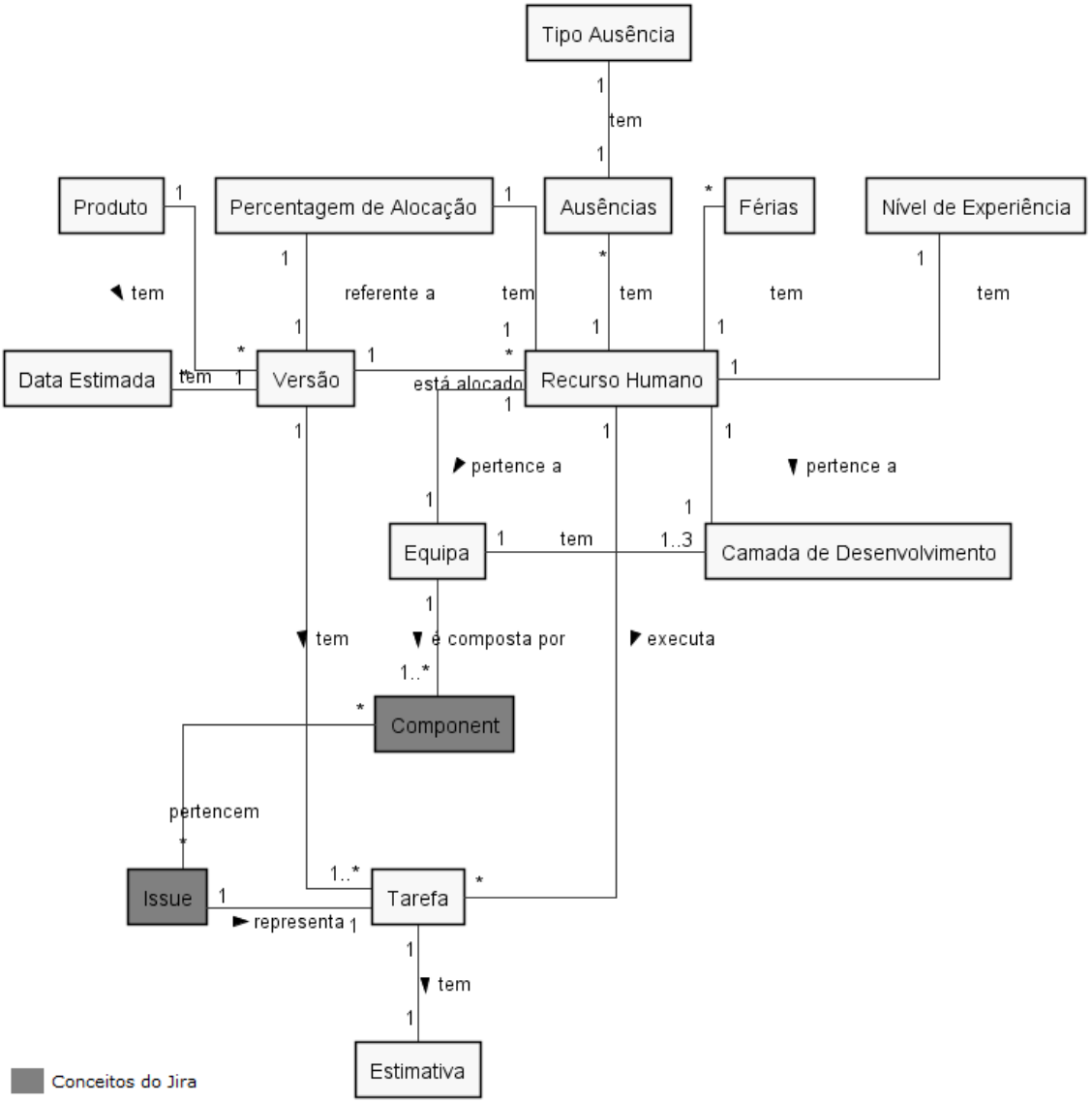

Figura 4.1: Modelo de Domínio

Os objetos do domínio da Figura 4.1 foram identificados a partir dos requisitos solicitados pela Equipa de *Release Planning* da ALERT, e são:

- **Produto: Produto** representa um dos produtos da *suite* da ALERT.

- **Versão: Versão** representa o conjunto de tarefas que devem ser realizadas para a *release* de uma nova versão de um determinado produto. Cada tarefa tem uma **Estimativa** que se traduz num custo estimado em horas para o seu desenvolvimento. Estas tarefas tipicamente estão capturadas em *issues* no Jira Software.
- **Data Estimada:** A **Data Estimada** corresponde à estimativa que o sistema calcula para o final do desenvolvimento da **Versão**.
- **Recurso Humano:** Os **Recursos Humanos** são colaboradores da empresa que estão alocados ao desenvolvimento da versão, cada recurso humano tem um **Nível de Experiência** e uma **Percentagem de Alocação** para cada uma das versões a que estão alocados.
- **Equipa: Equipa** representa as equipas de desenvolvimento da ALERT e podem estar divididas em três **Camadas de Desenvolvimento** (UI/UX, MW e DB). Para além disto, cada equipa tem associado a si um ou mais**Components**, que é um conceito do Jira Software.
- **Férias/Ausências:** Correspondem aos dias em que o **Recurso Humano** está indisponível para trabalhar, estes dados são importantes pois tem impacto no calculo da **Data Estimada**.

4.1.2 Modelo de Dados

Um Modelo de Dados é um modelo que descreve dados e as relações entre os dados. O modelo contém entidades que são apresentadas através dos dados que elas têm e não como se comportam ou quais são as suas responsabilidades. [44].

A Figura 4.1.2, apresenta o Modelo de Dados Relacional para o problema em análise.

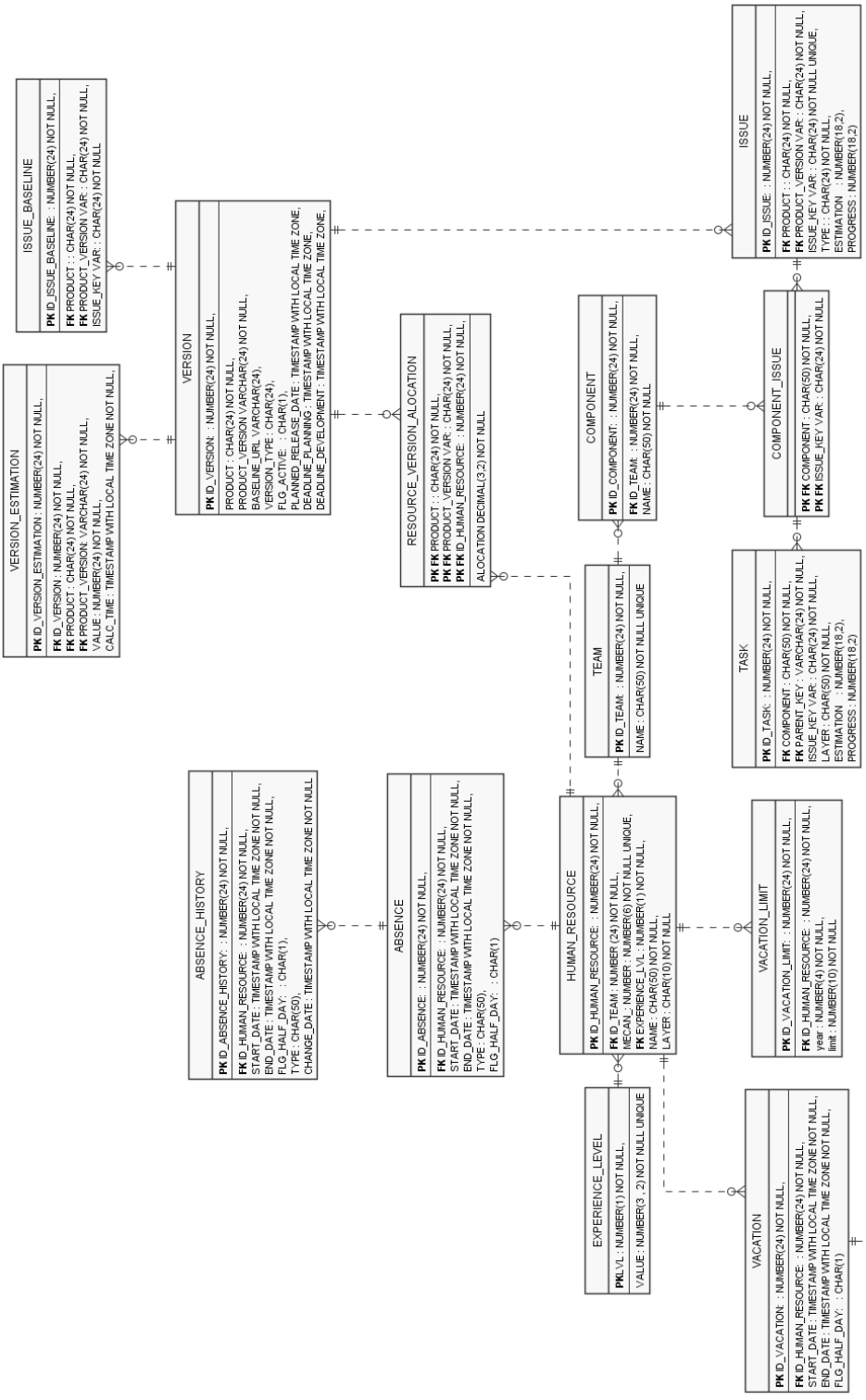

Figura 4.2: Modelo de Dados

Este modelo de dados é o reflexo dos conceitos capturados no modelo de domínio apresentado anteriormente. Analisando o modelo tem-se as seguintes tabelas do sistema:

- **VERSION:** Esta tabela contém toda a informação relacionada a uma versão, alguns do seus dados são provenientes de um *issue* do Jira.
- **VERSION_ESTIMATION:** Esta tabela contém todas as estimativas que foram calculadas para uma determinada versão de um determinado produto. É utilizada chave-composta PRODUCT PRODUCT_VERSION para identificar a qual versão é referente. Além de mais, esta chave é bastante utilizada nas diversas tabelas.
- **VACATION:** Esta tabela contém as datas de férias dos recursos humanos.
- **VACATION_LIMIT:** Esta tabela contém o limite de férias que cada recurso humano tem ao longo de um determinado ano.
- **VACATION_HISTORY:** Esta tabela contém o registo de alterações que foram ocorrendo a nível de troca de datas de férias.
- **TEAM:** Esta tabela identifica uma equipa de desenvolvimento da ALERT.
- **TASK:** Esta tabela corresponde a um *issue* do Jira do tipo *Estimate*. Nesta tabela está guardada a estimativa de uma determinada tarefa que compõe um *issue*.
- **RESOURCE_VERSION_ALOCATION:** Esta tabela guardas as percentagens de alocação de cada recurso humano às diversas versões.
- **ISSUE:** Esta tabela guarda os diversos *issues* provenientes do Jira. Os *issues* são compostos por diferentes *Tasks*.
- **ISSUE_BASELINE:** Esta tabela guarda a *baseline* de uma determinada versão. Sempre que uma *baseline* de uma versão é alterada são apagados os registos da tabela e preenchidos novos.
- **HUMAN_RESOURCE:** Esta tabela guarda as informações dos recursos humanos.
- **EXPERIENCE_LEVEL:** Esta tabela contém os diferentes níveis de experiência que um recurso humano pode assumir.
- **COMPONENT:** Esta tabela corresponde aos *components* do Jira.
- **COMPONENT_ISSUE:** Esta tablea relaciona os *issues* com os *components*.
- **ABSENCE:** Esta tabela contém as datas de ausências dos recursos humanos.
- **ABSENCE_HISTORY:** Esta tabela contém o registo de alterações que foram ocorrendo a nível de troca de datas de ausências.

4.2 Engenharia de Requisitos

Engenharia de Requisitos é uma disciplina da Engenharia de *Software* e resume-se ao uso sistemático e repetitivo de técnicas para obter, documentar e manter um conjunto de requisitos de um determinado *software* que atendem aos objetivos de um determinado negócio [45].

A Figura 4.3 apresenta as sub-disciplinas que são compõe a Engenharia de Requisitos.

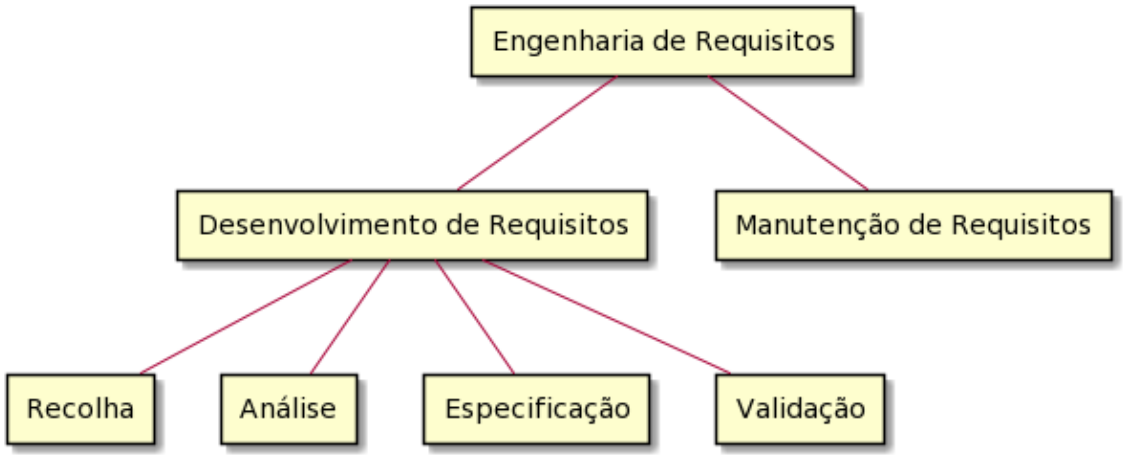

Figura 4.3: Sub-disciplinas da Engenharia de Requisitos de *Software* [adaptado de [46]]

Posto isto, esta secção visa documentar alguns artefactos resultantes da aplicação de engenharia de requisitos no projecto em apreço.

Primeiramente, foram identificadas as partes interessadas no projeto, seguindo-se da identificação dos atores do sistema. No final, em terceiro e quarto lugar surge a identificação dos requisitos do sistema de acordo com o sistema de classificação FURPS+.

O sistema FURPS+ é um sistema utilizado para a classificação de requisitos. O acrónimo FURPS corresponde, em inglês a, **F**unctionality, **U**sability, **R**eliability, **P**erformance, **S**upportability, que corresponde em português a Funcionalidade, Usabilidade, Fiabilidade/-Segurança, Desempenho e Suporte. O **+** que surge no final do acrónimo diz respeito a restrições que possam existir, como restrições de desenho, de implementação, de interface e de *hardware*.

Sendo que a categoria Funcionalidade diz respeito aos requisitos funcionais, todos os requisitos funcionais foram agrupados em 4.2.3. As restantes categorias dizem respeito aos requisitos não funcionais, que foram agrupados em 4.2.4.

4.2.1 Partes Interessadas

Partes interessadas, como o termo indica, é um conjunto de pessoas que possuem algum tipo de interesse no projeto em apreço. Tipicamente, as partes interessas são identificadas aquando a recolha de requisitos. Para este projeto foram identificadas as seguintes partes interessadas:

- **Equipa de Release Planning:** é a parte interessada principal pois será a que mais beneficiará do bom desempenho da plataforma; pretende que o tempo que dispõe para planear as *releases* seja reduzido e que o processo de monitorização seja simplificado pela centralização de diferentes indicadores numa única plataforma;
- **Equipa de Desenvolvimento:** é outro grupo de utilizadores que tem interesse nesta plataforma, principalmente os lideres das equipas, no acesso à consulta da monitorização das releases.

- **ALERT**: é a detentora dos produtos; possui o interesse de melhorar o planeamento para os seus produtos com o intuito de disponibilizar a melhor versão possível dos mesmos dentro dos prazos previstos; para além disso pretende que o processo de *Release Planning* melhore e que ocorra de forma mais rápida o que implicará a redução do tempo despendido a planear os produtos da ALERT;

4.2.2 Atores do Sistema

Aqui serão apresentados os atores que interagem com sistema em consideração nesta dissertação. Para que, deste modo, seja possível apresentar quais os utilizadores e as suas responsabilidades na plataforma.

Ator em Linguagem de Modelação Unificada, do inglês *Unified Modeling Language (UML)*, define um papel desempenhado por um utilizador ou outro sistema que interage com algum sistema [47].

Os atores em consideração para este sistema são:

- Membro da Equipa de *Release Planning* que, para além de poder acompanhar o monitorização das *releases* tem funcionalidades que só ele pode desempenhar.
- Membro da Equipa de Desenvolvimento que apenas irá ter acesso às funcionalidades de monitorização da *release*.
- O Jira *Software* é um ator na medida em despoleta algumas operações mediante eventos ocorridos no próprio Jira *Software*.

4.2.3 Requisitos Funcionais

Um Requisito Funcional, em Engenharia de Software, define uma funcionalidade de um sistema *software* ou do seu componente [45]. Por outras palavras, os requisitos funcionais representam as tarefas, ações ou atividades necessárias que devem ser realizadas pelo sistema.

A seguir, serão apresentados em detalhe três requisitos funcionais correspondentes a três das funcionalidades do sistema descrito neste documento. No Anexo B é possível consultar uma lista completa de todos os restantes requisitos funcionais deste projeto. Cada requisito funcional é acompanhado por um *Short Sequence Diagram (SDD)* que o ilustra. SDD é um diagrama em UML que apresenta as interações entre dois objetos organizadas pela sequência de tempo.

FR-01 - Carregar dados dos Recursos Humanos de um ficheiro Excel

Tabela 4.1: Detalhes do requisito FR-01

Descrição	Um membro da equipa de *Release Planning* deseja carregar no sistema um ficheiro Excel com as férias e ausências dos Recursos Humanos.
Prioridade	Alta

Simulação/Sequência de Respostas

Pré-condições:

- O membro da equipa deve ter permissões para efectuar o carregamento do ficheiro.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos.
4. O sistema apresenta a *dashboard*.
5. O membro navega na *dashboard* até encontrar a opção de carregar dados de férias/ausências.
6. O sistema apresenta uma área que permite ao utilizador seleccionar o ficheiro que deseja carregar.
7. O membro da equipa seleciona o ficheiro que deseja carregar e submete o ficheiro.
8. O sistema apresenta uma mensagem de sucesso.

Pós-condições:

- Os dados do ficheiro são validados pelo sistema.
- Os dados do ficheiro são persistidos numa base de dados.
- O sistema verifica se houve alteração aos dados previamente existentes no sistema, caso isso se verifique, o sistema atualiza os dados e persiste um histórico dos dados antigos.

Validações:

- As datas de férias/ausências não podem incluir fins-de-semana nem feriados;
- A data de inicio de um período de férias/ausências não pode ser após a data de fim.
- Os dias de férias de um determinado Recurso Humano não pode exceder o limite de férias que esse Recurso Humano tem para um determinado ano civil.

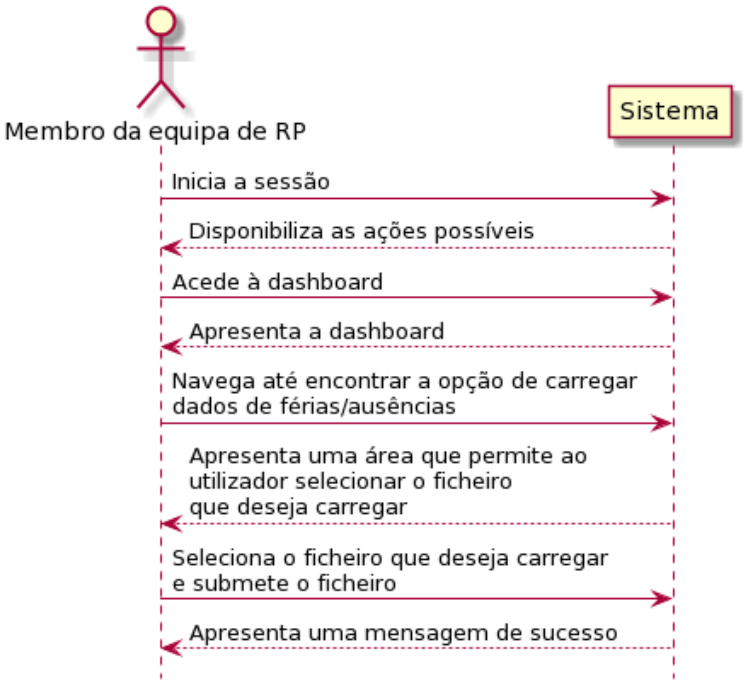

Figura 4.4: SSD FR-01

FR-02 - Listar os Recursos Humanos

Tabela 4.2: Detalhes do requisito FR-02

Descrição	Um membro da equipa de *Release Planning* deseja listar os Recursos Humanos que estão guardados no sistema.
Prioridade	Alta
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para efectuar a ação de listar os recursos humanos. • O sistema deve ter Recursos Humanos previamente inseridos. **Fluxo de Eventos:** 1. O membro da equipa inicia a sessão do sistema. 2. O sistema disponibiliza as ações possíveis. 3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos. 4. O sistema apresenta a *dashboard*. 5. O membro da equipa navega na *dashboard* até encontrar a opção de listar os Recursos Humanos guardados no sistema. 6. O sistema apresenta a lista dos Recursos Humanos com as suas informações mais relevantes.

Pós-condições:
- n/a

Validações:
- n/a

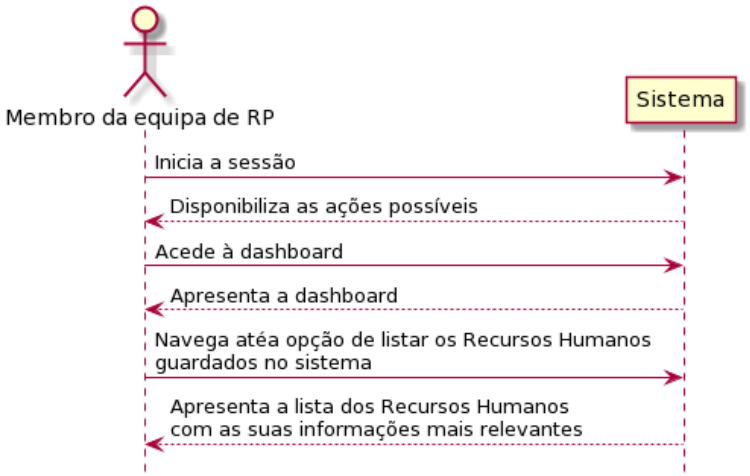

Figura 4.5: SSD FR-02

FR-03 - Alterar o nível de experiência de um Recurso Humano

Tabela 4.3: Detalhes do requisito FR-03

Descrição	Um membro da equipa de *Release Planning* altera o nível de experiência de um Recurso Humano guardado no sistema.
Prioridade	Alta
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para efectuar a ação de alterar o nível de experiência de um Recurso Humano. • O sistema deve ter Recursos Humanos previamente inseridos.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* até encontrar a opção de listar os Recursos Humanos guardados no sistema.
6. O sistema apresenta a lista dos Recursos Humanos com as suas informações mais relevantes.
7. O membro da equipa seleciona o Recurso Humano que deseja alterar o nível de experiência.
8. O sistema apresenta as ações que pode executar naquele Recurso Huamno.
9. O membro da equipa navega até à opção que lhe permite alterar o nível de expeiência.
10. O sistema apresenta os níveis de experiência possíveis.
11. O membro da equipa escolhe um nível e pressiona guardar.
12. O sistema questiona se o membro da equipa deseja confirmar a ação.
13. O membro da equipa confirma a ação.
14. O sistema atualiza o nível de experiência do Recurso Humano e informa sucesso.

Pós-condições:

- O Recurso Humano fica com um novo nível de experiência a si associado.

Validações:

- n/a

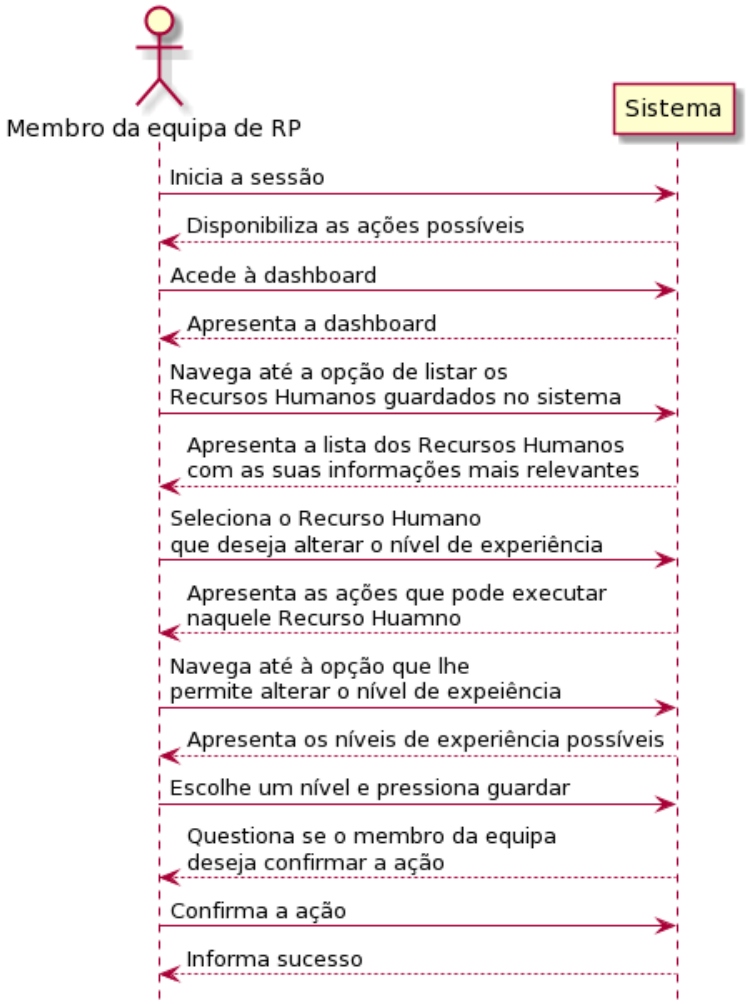

Figura 4.6: SSD FR-03

Sumário

Os requisitos funcionais acima apresentados traduzem as funcionalidades que o sistema deve apresentar para alcançar o seu propósito. O detalhe dos requisitos e as suas prioridades foram estabelecidos em concordância com a equipa de *Release Planning* da ALERT.

A Tabela 7.3 resume todos os requisitos funcionais especificados previamente.

Tabela 4.4: Tabela de Requisitos Funcionais

ID	Requisito Funcional	Prioridade
FR-01	Carregar dados dos Recursos Humanos de um ficheiro Excel	Alta
FR-02	Listar os Recursos Humanos	Alta
FR-03	Alterar o nível de experiência de um Recurso Humano	Alta
FR-04	Definir a percentagem de alocação de um Recurso Humano a uma versão	Alta
FR-05	Consultar as disponibilidades de um Recurso Humano	Média
FR-06	Criar uma versão	Alta
FR-07	Alterar uma versão	Alta

FR-08	Adicionar issue de uma versão	Alta
FR-09	Atualizar issue de uma versão	Alta
FR-10	Apagar issue de uma versão	Alta
FR-11	Listar versões	Alta
FR-12	Consultar os indicadores estatísticos de uma versão	Alta
FR-13	Fazer download de relatórios de uma versão	Média
FR-14	Consultar o histórico de uma versão	Média
FR-15	Apagar o histórico de uma versão	Baixa
FR-16	Consultar o histórico de férias/ausências de um Recurso Humano	Baixa
FR-17	Apagar o histórico de férias/ausências de um Recurso Humano	Baixa
FR-18	Login	Alta

A Figura 4.7 apresenta um diagrama UML de casos de uso.

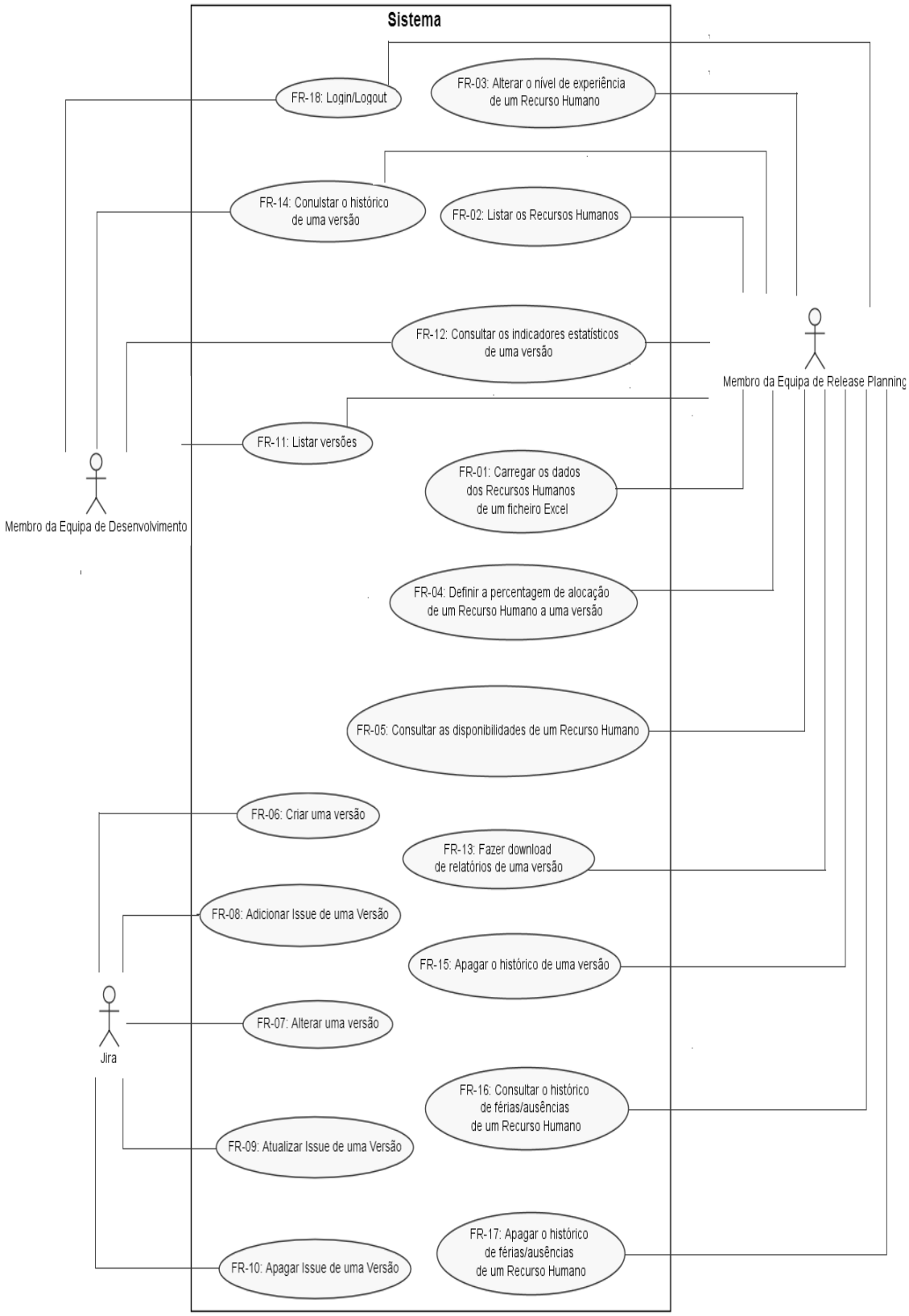

Figura 4.7: Diagrama de Casos de Uso

4.2.4 Requisitos Não Funcionais

Após a identificação dos requisitos funcionais que ajudam a interpretar o que a plataforma irá fazer, procedeu-se à identificação dos requisitos não funcionais.

Requisitos Não Funcionais, em Engenharia de Software, são requisitos relacionados com o uso do sistema tendo em consideração diversos aspectos [45].

Os requisitos não funcionais apresentados nesta sub-secção estão categorizados de acordo com o sistema FURPS+, introduzido na Secção 4.2.

Usabilidade

A usabilidade captura e avalia os requisitos com base na interface do usuário (por exemplo, acessibilidade, estética e consistência) [48]. No projeto em consideração, os diferentes requisitos não funcionais relacionados à usabilidade foram identificados:

- O sistema deve apresentar um elevado nível de usabilidade, a plataforma é uma ferramenta de suporte para o desenvolvimento de produto da ALERT e deve proporcionar aos seus utilizadores facilidade e rapidez na execução das suas tarefas;
- O sistema deve ser de fácil compreensão para quem o utiliza de modo a que não haja necessidade de formação de suporte ao seu uso;
- A sua interface deve ser intuitiva, deve permitir fácil memorização para a repetição de atividades de forma rápida.

Fiabilidade/Segurança

As restrições de fiabilidade/segurança incluem aspetos como disponibilidade, precisão, capacidade de recuperação e tempo médio entre falhas [48]. Assim, no âmbito deste projeto identificaram-se os seguintes requisitos de fiabilidade e segurança:

- O lado do servidor deve permanecer disponível e funcional após falhas;
- O sistema deve estar acessível vinte e quatro horas por dia, sete dias por semana;

Desempenho

Os requisitos de desempenho estão relacionados com o desempenho do sistema, isto é, quão bem este executa determinadas funções sob condições específicas [48]. Os requisitos de desempenho impostos pela equipa de *Release Planning* da ALERT foram:

- O tempo médio de resposta dos *endpoints* do serviço Representational State Transfer (REST) deve ser inferior a 1 segundo;
- A determinação da data estimada para o fim do desenvolvimento de uma determinada *release* não deve ser superior a 30 segundos;
- Tempo médio do carregamento das páginas de interface *web* inferior a 5 segundos;
- O sistema deverá suportar múltiplos acessos concorrentes sem que coloque em causa a produtividade dos utilizadores.

Suporte

As restrições de capacidade de suporte estão essencialmente relacionadas a limitações de linguagem, adaptabilidade, manutenção, compatibilidade e escalabilidade [48]. No âmbito deste projeto, foram capturados os seguintes requisitos relacionados com o suporte do sistema:

- O sistema deve ter capacidade de expansão e permitir a adição de novos módulos, se e quando necessário;
- O sistema deve poder ser acedido através de qualquer um dos principais navegadores de Internet (Google Chrome, Safari, Mozilla Firefox, Opera).

Restrições de Desenho

Uma restrição de desenho limita a forma como o sistema é delineado (por exemplo, o uso de uma base de dados relacional é necessário) [48]. Para o desenvolvimento do protótipo em questão, as restrições relacionadas ao desenho da solução foram:

- O desenho da solução deve ir de encontro com as práticas de desenvolvimento de *software* aplicado na ALERT.
- Durante o desenho do projeto, devem ser adoptados bons princípios e padrões de engenharia de *software*.

Restrições de Implementação

Este tipo de restrição afeta a forma como o sistema é construído e também pode impactar o desenho do sistema [48]. Para o projeto em apreço, foram identificadas as seguintes restrições de implementação:

- O sistema deve ser desenvolvido de acordo com a *stack* tecnológica (Javascript, Angular, HTLM5, JAVA E PL/SQL, Base de dados Oracle) da ALERT;
- O sistema deve suportar o protocolo REST para comunicações com a Application Programming Interface (API) do Jira *Software*;
- O sistema deve ser capaz de consultar a base de dados do Jira;
- Durante o desenvolvimento do projeto, devem ser usados bons princípios de codificação.

Restrições de Interface

Restrições ao nível da interface são aquelas que a afetam a forma como a comunicação é feita com outros componentes. Um tipo de restrição de interface é, por exemplo, restringir o protocolo de comunicação [48]. Identificaram-se as seguintes restrições:

- A interface entre o Jira *Software* e o sistema é feita através do protocolo REST.
- A interface entre a base de dados do Jira *Software* e o sistema é feita através API JDBC.

4.3 Análise ao Jira Software

Sendo que um dos objetivos do projeto em apreço é a integração da plataforma a desenvolver com o Jira *Software*, esta secção apresenta uma análise ao Jira *Software*.

Primeiramente, será apresentada uma análise ao Jira *Software*, analisando aspetos como os seus termos-chave, as opções de integração que oferece, a base de dados e o esquema de dados, seguindo-se de uma introdução à estrutura do Jira na ALERT.

4.3.1 Jira Software

Como mencionado ao longo desta dissertação, o Jira *Software* [49] é uma ferramenta de desenvolvimento de *Software* da Altassian utilizada por equipas ágeis.

Figura 4.8: Jira Software Logótipo [Fonte:[49]]

O Jira *Software* é um dos produtos que compõe a plataforma Jira que foi projetada para ajudar equipas a planear, atribuir, monitorizar e gerir o trabalho e que oferece outros produtos como Jira *Service Managment*, Jira *Core* e Jira *Align*.

O Jira *Software*, objeto de estudo no âmbito desta dissertação, é direcionado para o planeamento ágil de desenvolvimento de *software* sendo capaz de lidar com rastreamento de *bugs*, gerenciamento de projetos e produtos, entre outras coisas.

Em termos de utilização, as empresas podem optar por utilizar o Jira *Software Cloud* como *Software as a Service (SaaS)* ou, então, optar pelo Jira *Software Data Center* como *On-Premise Software*, instalando o Jira no *hardware* da empresa ou em fornecedores de infraestruturas como o Azure e o AWS.

Figura 4.9: Ilustração das Opções de Hospedagem para o Jira *Software* [Adaptado de: [49]]

Termos-chave do Jira Software

Em seguida, na Tabela 4.5 são apresentados os termos-chave associados ao Jira *Software*.

Tabela 4.5: Termos-chave do Jira *Software*

Termo	Definição
Issue	Refere-se a uma tarefa única de trabalho de qualquer tipo ou tamanho que é rastreado desde a sua criação até à conclusão.
Projeto	Conjunto de *issues* que tem um propósito ou contexto comum. Os *issues* agrupados em projetos podem ser configurados de várias maneiras.
Component	Contentores de *issues* que permitem dar alguma estrutura aos projetos, dividindo-os em funcionalidades, equipas, subprojectos, etc...
Board	É parte de um projeto que exibes *issues*, permitindo, de forma fléxivel, visualizar e gerir o trabalho em curso. Por outras palavras, é ma representação visual do fluxo de trabalho de uma equipa dentro de um projeto.
Workflow	Representam o caminho sequencial de um *issue* desde a criação até à conclusão. A Figura 4.10 ilustra um *workflow* básico, porém os *workflows* são configuráveis no Jira.
Agile	Apesar de não ser um termo específico do Jira *Software*. Jira *Software* tem conjuntos específicos de requisitos projetados especialmente para a Metodologia Agil, incluindo *Scrum* ou *Kanban*. O Jira *Software* não torna as equipas em equipas ágeis mas é uma ferramenta construída com o propósito de ajudar as equipas a lá chegar.

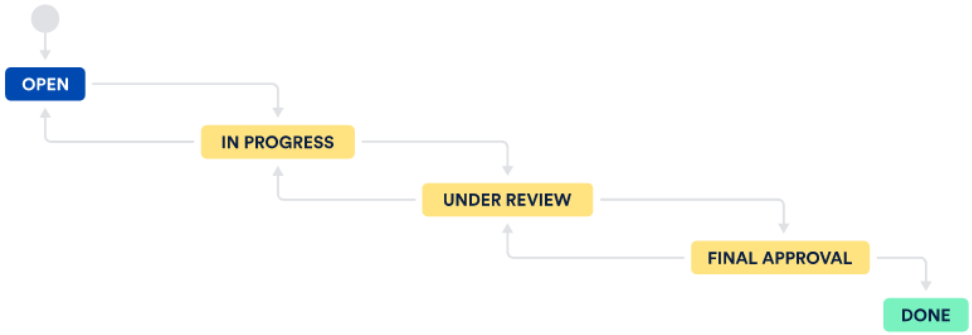

Figura 4.10: Ilustração de um *Workflow* Básico do Jira [Fonte : [49]]

4.3.2 Integrações com o Jira Server

Com vista a integrar a plataforma em apreço com o Jira *Software* da ALERT procedeu-se a uma análise relativa aos métodos de integração que o Jira *Software* oferece. A ALERT tem o Jira *Software* hospedado localmente, pelo que consideraram-se os seguintes métodos de integração [50]:

- Jira *Software Server REST API*
- Jira *Software Server Java API*
- *WebHooks*

Em seguida é apresentada uma análise a cada uma das opções.

Jira Software Server REST API

O Jira *Software* fornece uma API REST[1] que disponibiliza um conjunto de recursos[2] que permitem desenvolver *add-ons* para o Jira, integrações com o Jira e outras aplicações, ou simplesmente, interações entre *scripts* e o Jira.

O acesso aos recursos do Jira é através de caminhos Uniform Resource Identifier (URI). Tipicamente, estes URIs tem a seguinte estrutura: **http://[HOST]:[PORT]/context/rest/api-name/api-version/resource-name**.

A troca de mensagens entre o Jira e os sistemas externos ocorre sobre o protocolo HTTP e as essas mensagens estão formatadas em JavaScript Object Notation (JSON).

O seguinte extracto de código (Listagem 4.1) representa uma resposta em JSON de um pedido à REST API do Jira.

```
{
    "self": "http://www.example.com/jira/rest/api/2/issue/10010/worklog/10000",
    "author": {
        "self": "http://www.example.com/jira/rest/api/2/user?username=fred",
        "name": "fred",
        "displayName": "Fred F. User",
        "active": false
    },
    "updateAuthor": {
        "self": "http://www.example.com/jira/rest/api/2/user?username=fred",
        "name": "fred",
        "displayName": "Fred F. User",
        "active": false
    },
    "comment": "I did some work here.",
    "updated": "2017-12-07T09:23:19.553+0000",
    "visibility": {
        "type": "group",
        "value": "jira-developers"
    },
    "started": "2017-12-07T09:23:19.553+0000",
    "timeSpent": "3h 20m",
    "timeSpentSeconds": 12000,
    "id": "100028",
    "issueId": "10002"
}
```

Listing 4.1: Exemplo de uma Resposta JSON da REST API do Jira

[1] https://docs.atlassian.com/software/jira/docs/api/REST/7.6.1/
[2] https://docs.atlassian.com/software/jira/docs/api/REST/7.6.1/#api/2/

Jira Software Server Java API

Para além da API REST supramencionada, o Jira oferece uma Java API[3] direcionada para quem deseja desenvolver Plugins2 que, em alternativa ao uso API REST, permite que sejam desenhados e construídos *plugins* em Java que estendem funcionalidades do Jira *Software*, que podem ser, uma vez mais, integrações com serviços externos, novas funcionalidades para o Jira *Software* ou um novo produto de *software* executado no Jira *Software*.

O *plugin* ou *add-on* é, então, um conjunto de código que pode ser instalado no Jira para lhe adicionar novas funcionalidades ou alterar o comportamento de funcionalidades pré-existentes.

Há dois grandes tipos de *plugins*:

- **Aplicações de Sistema**, que podem ser adicionadas diretamente ao Jira para fornecer funções básicas.
- **Aplicações Personalizadas e de Terceiros**, que são aplicações desenvolvidas especificamente para uma determinada instalação do Jira e que se integram com o Jira.

WebHooks

Em desenvolvimento *web*, segundo o modelo cliente-servidor[4], um *WebHook* é um retorno de uma chamada através de HTTP definida pelo cliente.

Deste modo, Jira *WebHooks* [51] podem ser utilizados para notificar uma aplicação externa quando certos eventos ocorrem no Jira.

A quando o registo de um novo *WebHook* no Jira a seguinte informação é necessária:

- Um nome para o *WebHook*;
- Um *Uniform Resource Locator (URL)* para onde o *WebHook* deverá enviar a chamada de retorno;
- O *scope* do *WebHook*, ou seja, o cojunto de *issues* que compõe o *WebHook*. O *scope* pode ser "todos os *issues* ou um conjunto de *issues* especificados com JQL;
- O tipo de eventos do Jira que devem despoletar a chamada de retorno.

Um *WebHook* assemelha-se a algo deste género:

- **Name:** "Exemplo *Webhook*"
- **URL:** www.3rd-party-app.com/webhookreceiver
- **Scope:** issue = Task
- **Events:** Issue Updated, Issue Created

Para registar *WebHook* novos há duas alternativas: utilizando a consola de administrador do Jira ou utilizando a Jira REST API.

[3] https://docs.atlassian.com/software/jira/docs/api/7.6.1/

[4] a nível do software, cliente-servidor descreve a relação entre dois programas de computador no qual um deles, o cliente, realiza um pedido para outro programa, o servidor, que serve esse pedido

4.3.3 Base de Dados do Jira

O Jira utiliza uma Base de Dados Relacional para persistir os seus dados. Por defeito, ao configurar o Jira, uma nova conexão de base de dados com um motor H2 interno é gerada, opcionalmente pode-se optar por uma base de dados externa. A ALERT faz uso do PostgreSQL como motor de base de dados para a base de dados do seu Jira.

Dado que, um dos objetivos propostos é recolher indicadores estatísticos das *releases* e que uma das opções em consideração para atingir esse objectivo seria consultar a base de dados do Jira foi necessário analisar, não só o motor de base de dados utilzado pela ALERT para o seu Jira, mas também analisar o esquema de dados do Jira[5].

Esta análise facilita o trabalho de desenvolvimento da plataforma em consideração na medida em que permite perceber qual é o motor de base dados e como se pode efectuar uma ligação a ele e, também, como estão organizados os dados do Jira para poder ser mais simples e rápido consultar os dados persistidos.

4.3.4 Jira Software na ALERT

O Jira *Software* permite aos seus utilizadores usufruírem das suas potencialidades da forma que mais se adequem às suas necessidades.

Dado que, a ALERT utiliza o Jira de uma forma *sui generis*, em seguida será apresentada uma visão geral de como está estruturado o Jira na ALERT para contextualizar o leitor.

A ALERT criou para cada linha do seu produto ALERT® um projeto no Jira que reúne todos os *issues* relativos a essa linha do produto.

Esses projetos por sua vez podem ter diversos componentes, que dividem os *issues* do projeto pelas equipas de desenvolvimento, e versões, que dividem os *issues* pelas versões de desenvolvimento a que estão associados.

Cada *issue* tem um *issue type* que pode ser standard do Jira ou customizado. No caso da ALERT, quando um *issue* é do tipo "nova funcionalidade", a ele são associadas *subtasks* para estimativa de esforço por componente e camada de desenvolvimento.

A Figura 4.11 apresenta a estrutura do Jira da ALERT.

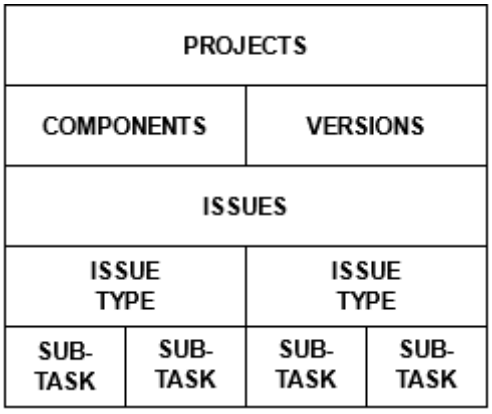

Figura 4.11: Estrutura do Jira da ALERT

[5]https://developer.atlassian.com/server/jira/platform/attachments/jira-7-9-2-database-schema.pdf

4.3.5 Sumário

Em suma, o Jira *Software* é uma ferramenta que visa apoiar o processo de desenvolvimento de *software*. Esta ferramenta é dirigida a equipas ágeis e tem alguns conceitos-chave que a caracteriza. É importante também realçar, que esta é uma ferramenta que oferece aos seus utilizadores facilidade de integração com outras aplicações, sendo que as opções de desenvolvimento de *software* em torno do Jira são vastas. No final, é apresentada a estrutura da ferramenta no contexto de desenvolvimento da ALERT.

Capítulo 5

Arquitetura da Solução

A arquitetura de *software* de um sistema corresponde ao conjunto de estruturas fundamentais de um sistema de *software*, que compreende os elementos de *software*, as relações entre eles e as propriedades de ambos [52].

Assim sendo, a arquitetura de *software* tem como objetivo estabelecer uma ponte entre a abstração dos objetivos do sistema e o sistema concreto. Uma arquitetura de *software* pode ser desenhada, analisada, documentada e implementada utilizando técnicas que fornecem suporte para atingir os objetivos abstratos e complexos definidos para o sistema, tornando a definição do sistema mais tangível [52].

Dito isto, este capitulo tem dois objetivos: o primeiro objetivo passa por apresentar algumas alternativas consideradas para o desenho da arquitetura e, em seguida, apresentar a Arquitetura de *Software* final com vista a ilustrar o sistema do projeto aqui documentado.

5.1 Apresentação das Alternativas

Esta secção tem como objetivo apresentar as alternativas desenhadas para o sistema de *software* descrito ao longo deste documento.

5.1.1 Vistas Lógicas

Tipicamente, a definição de uma vista lógica de um terminado sistema é através de diagramas de componentes. Diagramas de Componentes são diagramas UML que ilustram o relacionamento entre os diferentes componentes de um sistema. Componente, em UML, refere-se a um módulo de classes que representa sistemas ou subsistemas independentes com capacidade de interagir com o restante do sistema. A seguir, serão apresentadas as três alternativas consideradas para o problema analisado ao longo desta dissertação.

Alternativa 1

Na Figura 5.1 está ilustrada a primeira alternativa para a estruturação do sistema. Nesse diagrama está representada a parte do *Jira Software*, bem como outros componentes. Os componentes Web Portal, Release Monitor e Release Planner e base de dados Data Base visam ilustrar uma possibilidade de estruturação para o sistema.

O componente Jira Integrator é um componente que tem como objetivo estabelecer a integração entre o sistema e o Jira *Software*.

Nesta opção, o *backend* do sistema está segregado em dois componentes distintos - Release Monitor e Release Plan Support Tool - com responsabilidades bem definidas.

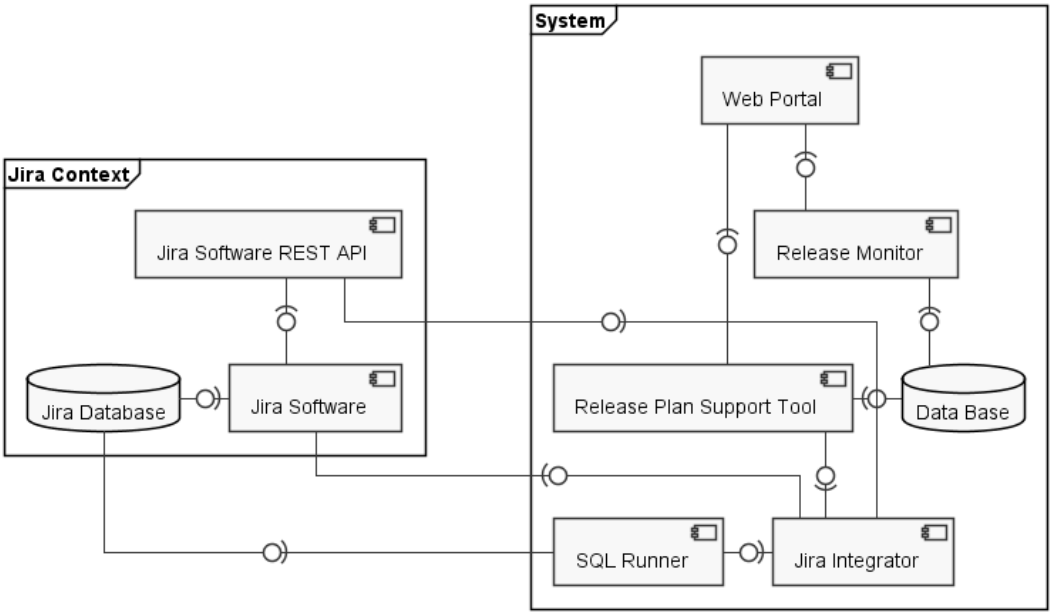

Figura 5.1: Diagrama de Componentes 1

É possível consultar a responsabilidade de cada componente na Tabela 5.1.

Tabela 5.1: Alternativa 1 - Responsabilidades do Componentes

Componente	**Responsabilidade**
Web Portal	Expor uma interface gráfica para o utilizador.
Release Plan Support Tool	Realizar os cálculos relativos à estimativa para o final do desenvolvimento das *release*; Regras relativas à alocação de recursos.
Release Monitor	Recolher todos os indicadores das diversas *releases*.
Data Base	Persistência de dados
Jira Integrator	Gestão dos *webhooks* enviados pelo Jira.
SQL Runner	Executar *queries* à base de dados do Jira.

Alternativa 2

Na Figura 5.2 está ilustrada a segunda alternativa para a estruturação do sistema. Este diagrama, à semelhança do anterior, ilustra a parte do *Jira Software*, assim como os restantes componentes.

Os componentes Web Portal, Release Tool e base de dados Data Base ilustram uma alternativa para a estruturação do sistema.

Nesta opção, o *backend* do sistema é composto por um componente apenas, o Release Tool.

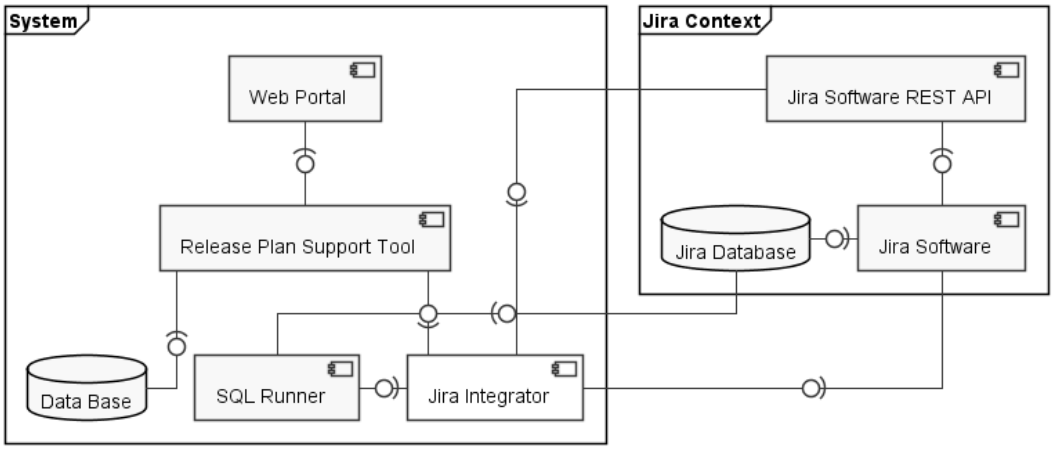

Figura 5.2: Diagrama de Componentes 2

A Tabela 5.2 apresenta as responsabilidades de cada componente acima desenhado.

Tabela 5.2: Alternativa 2 - Responsabilidades do Componentes

Componente	**Responsabilidade**
Web Portal	Expor uma interface gráfica para o utilizador.
Release Plan Support Tool	Realizar os cálculos relativos à estimativa para o final do desenvolvimento das *release*; Regras relativas à alocação de recursos; Recolher todas os indicadores das diversas *releases*.
Data Base	Persistência de dados
Jira Integrator	Gestão dos *webhooks* enviados pelo Jira.
SQL Runner	Executar *queries* à base de dados do Jira.

Alternativa 3

Por fim, a Figura 5.3 ilustra a terceira alternativa considerada. Esta alternativa tem por base a alternativa apresenta na Alternativa 1 e que apenas difere dessa na adição de um componente que serve de "ponte"entre o módulo responsável pela interface entre utilizador e sistema e o *backend* do sistema.

Este componente é o Gateway e tem como responsabilidade a gestão das APIs. Assim, é da sua responsabilidade a mediação entre o cliente (Web Portal) e os serviços disponibilizados no *backend*.

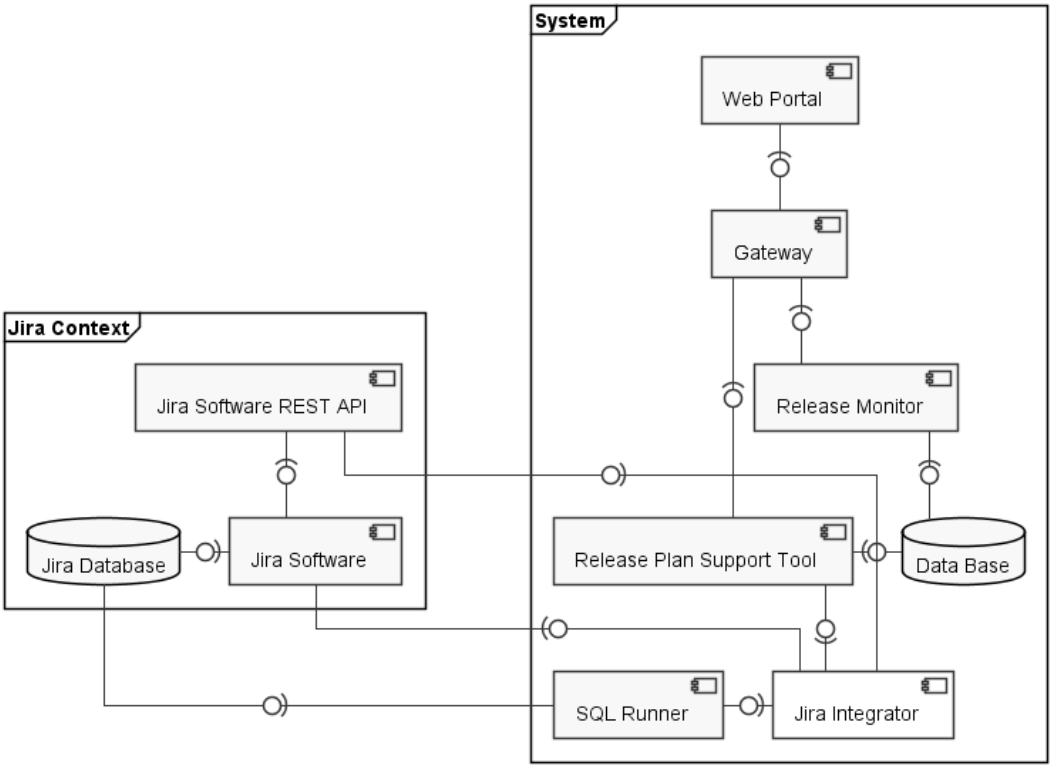

Figura 5.3: Diagrama de Componentes 3

Sumário

Anteriormente foram apresentados três possíveis desenhos para o sistema. A principal diferença é relativa *backend* da plataforma. A primeira é a segregação do *backend* em dois componentes com responsabilidades bem definidas, sendo que a segundo hipótese é a criação de um componente só responsável por todas as operações da plataforma. No final, surge a hipótese de, para além de segregar do *backend* em dois componentes, criar um único ponto de entreda no *backend*, através da criação de um componente com a função de *gateway*.

5.1.2 Vistas de Implantação

Associadas às vistas de implantação surgem os diagramas de implantação. Um Diagrama de Implantação, em UML, é um diagrama estrutural que descreve o aspecto do sistema em si. Isto é, o diagrama de implantação ilustra a implementação física dos componentes de *software* em componentes de *hardware*.

A seguir, serão apresentadas as duas alternativas consideradas para a implantação do sistema descrito nesta dissertação.[1]

[1]Para este desenho considerou-se a segunda alternativa do diagrama de componentes para simplificar a interpretação. Porém, a lógica de implementação é a mesma independente a alternativa que se venha a adotar.

Alternativa 1

No diagrama da Figura 5.4, para além do servidor que hospeda o Jira *Software* da ALERT (Jira Server) são apresentados outros dois servidores. O Jira Integrator Server, é um servidor pensado para hospedar todos os componentes responsáveis pela integração entre a plataforma e o Jira. Por fim, o servidor Server hospeda todos os componentes que constituem a plataforma em si.

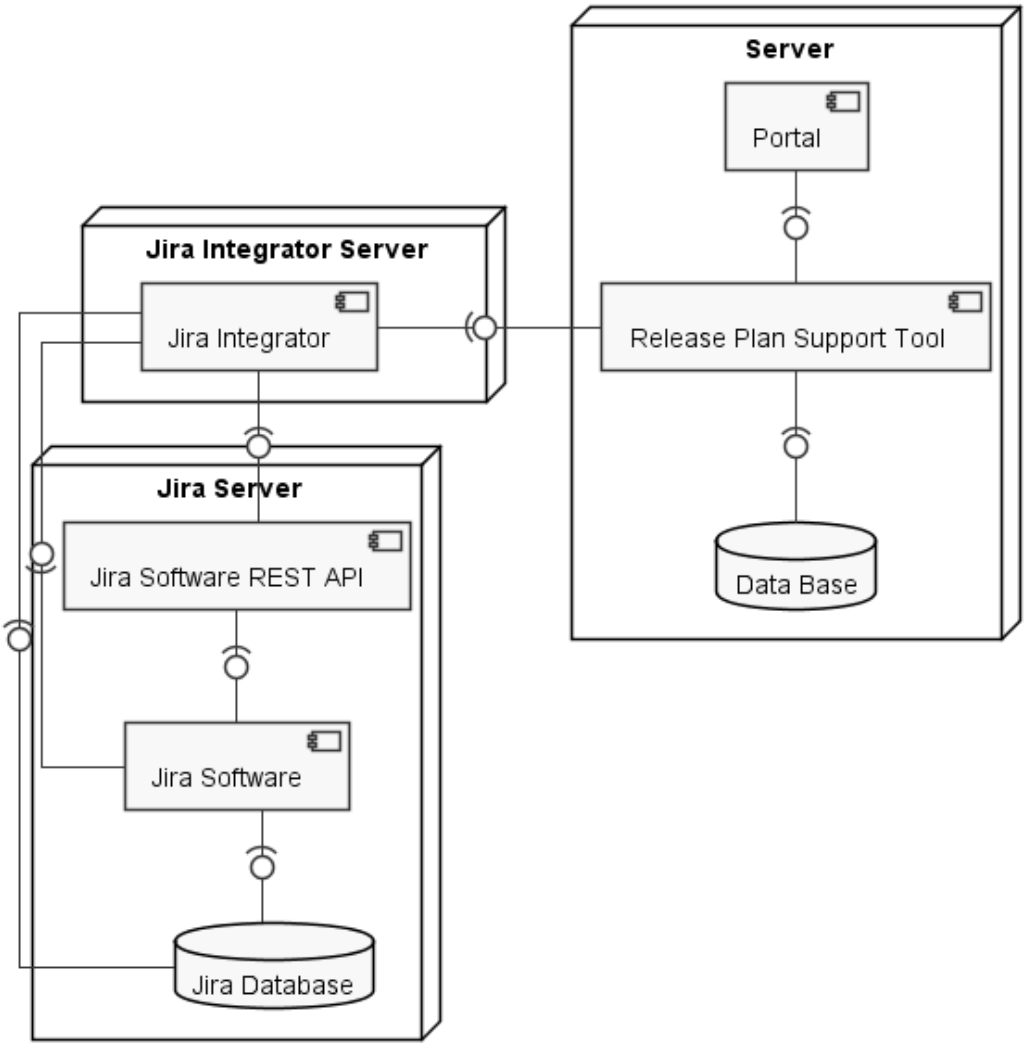

Figura 5.4: Diagrama de Implantação 1

Alternativa 2

A Figura 5.5 apresenta um diagrama de implantação. Este diagrama, bem como o da alternativa anterior, apresenta o servidor que hospeda o Jira *Software* da ALERT (Jira Server), porém, apenas apresenta um servidor (Server) para hospedar quer os componentes da plataforma, quer os componentes responsáveis pela integração da plataforma com o Jira *Software*.

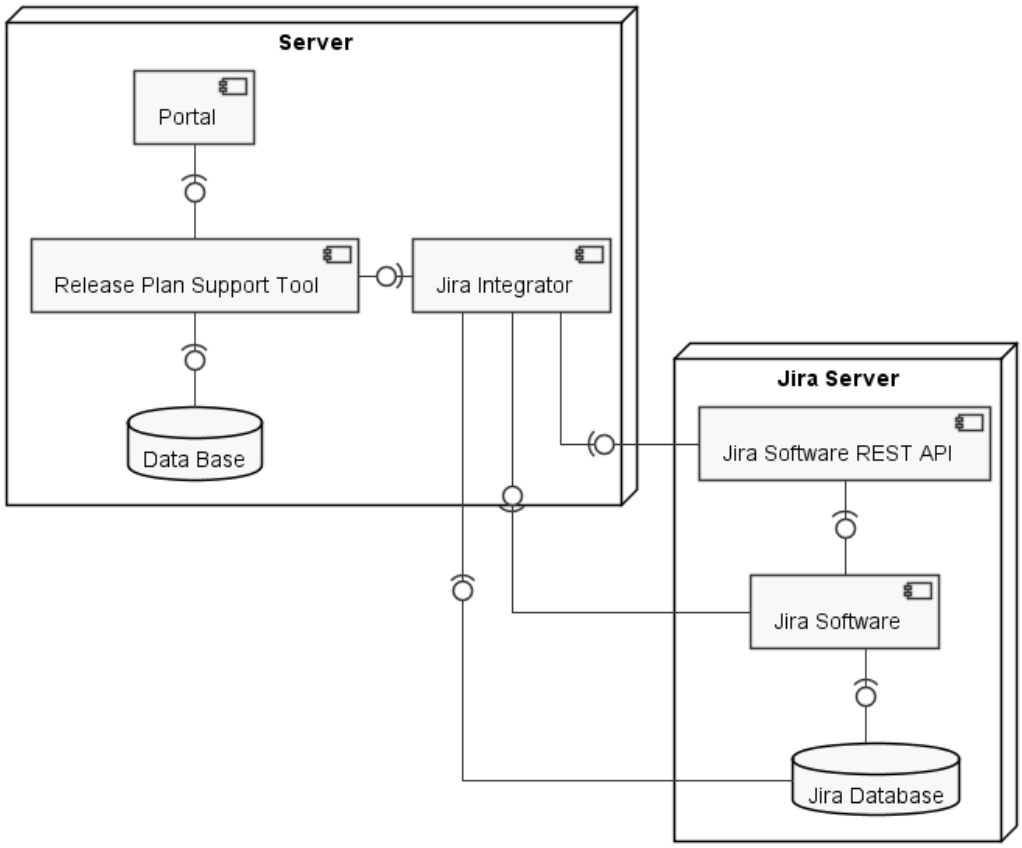

Figura 5.5: Diagrama de Implantação 2

Sumário

Acima foram apresentadas duas opções de como podem ser implantados os componentes que compõe a solução. A diferença entre as alternativas é que uma pressupõe que haja um servidor exclusivo para a hospedagem dos componentes de integração entre o Jira e a plataforma e a outra pressupõe que todos os componentes da plataforma e de integração são hospedados no mesmo servidor.

5.2 Arquitetura Final

Esta secção tem como objetivo descrever e detalhar a arquitetura pensada para construir a solução que dá resposta ao problema em apreço neste projeto.

Mediante as alternativas apresentadas na Secção 5.1, e, no que diz respeito ao desenho da solução, a escolha recai na Alternativa 3. Tendo, em consideração todos os requisitos definidos na Secção 4.2.4 e as boas práticas de desenho e programação, a escolha da alternativa 3 permite que se construa um *software* de fácil manutenção e evolução, permitindo que a adição de novas funcionalidades seja mais fácil. Quanto à implantação, e dado que o módulo de integração com o Jira, para já, não tem outro objetivo a não ser estabelecer a comunicação entre o Jira *software* e este sistema optou-se por implantar tudo numa máquina só.

Posto isto, para realizar esta descrição optou-se por utilizar o Modelo 4+1 de Arquitectura de *Software* pelo facto de ser um possível de ser possível ter várias vistas simultâneas da solução [53].

5.2.1 Modelo 4+1

O Modelo 4+1 (Figura 5.6) pretende descrever a arquitetura de *software*, e as suas decisões, organizadas entre cinco vistas [53]:

- **Vista Lógica:** Visa fornecer uma base para a compreensão da estrutura e organização do sistema, ilustrando os principais componentes e seus relacionamentos, que abrangem comportamentos arquiteturais significativos.
- **Vista de Implementação:** Ilustra o sistema na perspectiva do programador.
- **Vista de Processos:** Ilustra os processos do sistema e como eles comunicam com foco no tempo de execução do sistema.
- **Vista de Implantação:** Visa fornecer detalhes do sistema tendo em consideração a camada física, nestas vistas geralmente são apresentadas as conexões físicas entre os componentes do sistema.
- **Vista de Cenários:** Visa ilustrar as quatro vistas anteriormente definidas, para isso são apresentados alguns cenários (ou casos de uso).

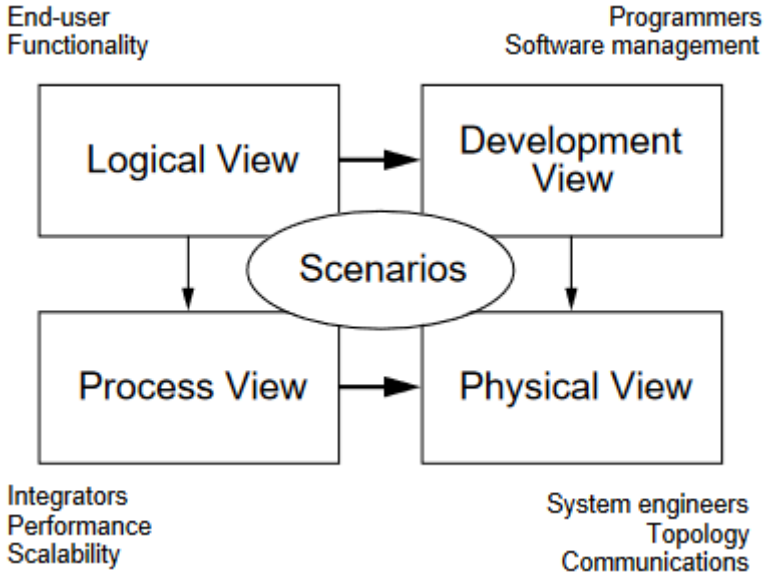

Figura 5.6: Modelo 4+1 [Fonte:[53]]

Posto isto, em seguida será apresentadas as vistas -Vista Lógica, Vista de Implementação, Vista de Processos, Vista de Implantação ilustradas por alguns casos de uso, ou cenários, que se tornam a quinta vista- que ilustram a arquitetura deste projeto.

5.2.2 Vista Lógica

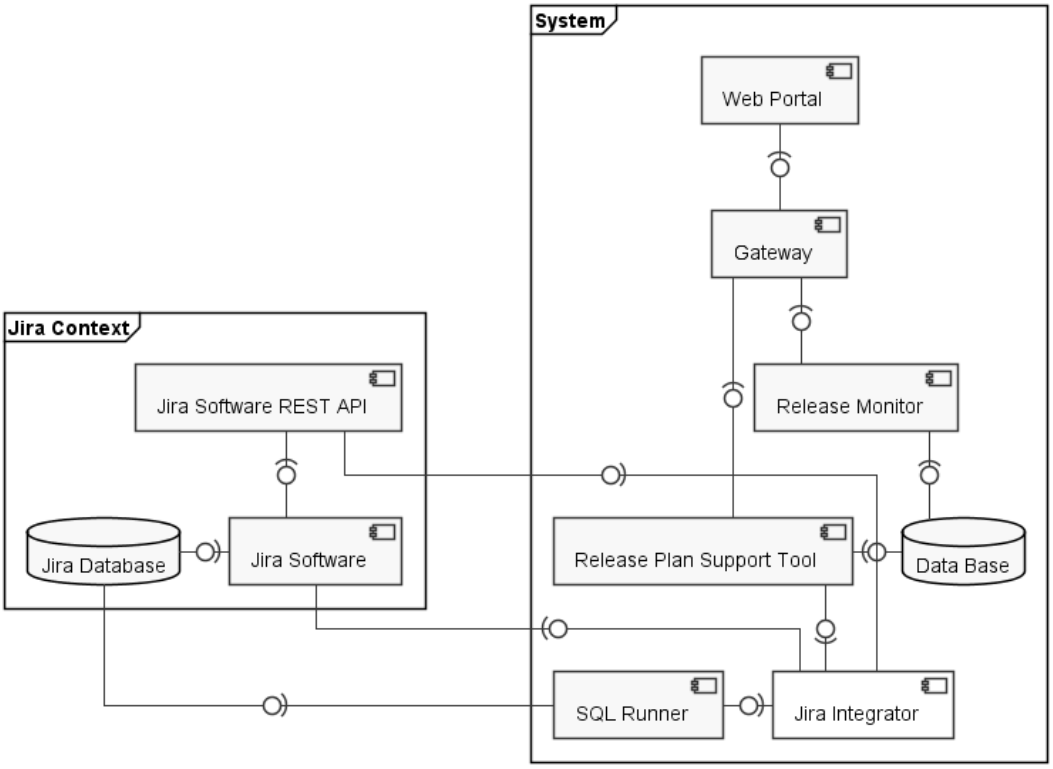

Figura 5.7: Diagrama de Componentes do Sistema

A Figura 5.7 apresenta o diagrama de componentes do sistema para o problema em apreço. Após uma análise identificam-se os principais componentes. Sendo estes,

- **Web Portal:** Este componente é a interface entre o sistema e o utilizador, sendo o componente responsável por apresentar uma interface gráfica ao utilizador do sistema.
- **Gateway:** Componente mediador que faz a ligação entre o cliente (Web Portal) e os serviços disponibilizados no *backend* (Release Monitor e Release Plan Support Tool).
- **Release Monitor:** Componente responsável por disponibilizar os diversos indicadores estatísticos das diversas versões que se encontram na base de dados.
- **Release Plan Support Tool:** Local onde é definida a lógica do projeto.
- **Jira Integrator:** Modulo que integra o sistema com o Jira *Software*. Este componente para além de "ficar à escuta"de eventos ocorridos na plataforma Jira, executa algumas operações no Jira através da API Jira.
- **SQL Runner:** Componente responsável por executar consultas à base de dados do Jira.

Posto isto, e de modo a esclarecer como foram desenhados os componentes Jira Integrator, Release Plan Support Tool e Release Monitor é apresentado, em seguida, uma explicação.

Jira Integrator

Este componente de integração foi desenhado de acordo com o diagrama de componentes da Figura 5.8.

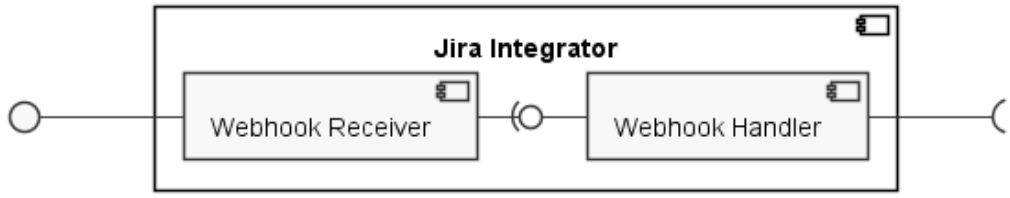

Figura 5.8: Diagrama de Componentes do Jira Integrator

Neste diagrama é possível perceber que o componente está dividido em dois componentes, para que, desse modo, cada componente tenha as suas responsabilidades bem definidas. Assim sendo, o Webhook Receiver tem como responsabilidade expor *endpoins* REST para receber os eventos despoletados pelo Jira *Software*. Por sua vez, o Webhook Hanlder tem como responsabilidade processar o evento e encaminha-lo para o seu destino.

Release Plan Support Tool e Release Monitor

No caso dos componentes Release Plan Support Tool e Release Monitor ambos seguem uma estrutura semelhante. Ambos foram desenhados para ter 3 componentes de *middleware* e um componente (ou vários) componentes responsáveis pela lógica de negócio.

Os componentes de *middleware* - Controllers, Services, Core - são comuns aos dois componentes e tem como objectivo expor a lógica do domínio sob a forma de um serviço *REST*.

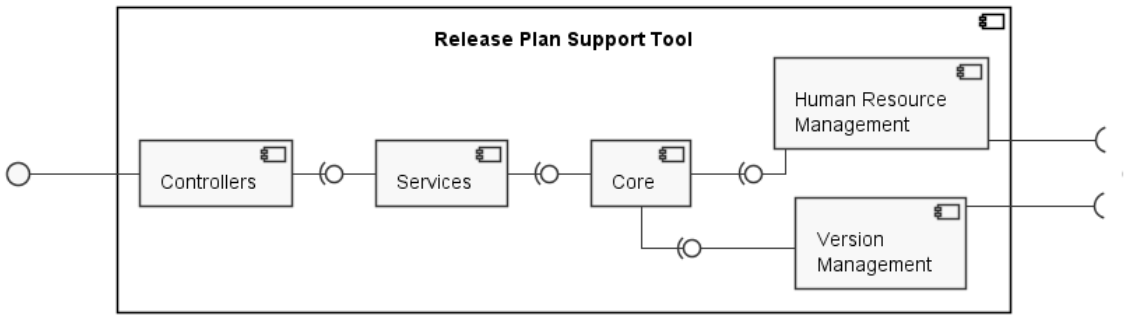

Figura 5.9: Diagrama de Componentes do Jira Integrator

Na Figura 5.9 destaque para o Human Resource Management responsável por lidar com lógica relativa aos recursos humanos envolvidos no desenvolvimento das *release* e para o Version Management responsável pela lógica relativa a gestão das versões.

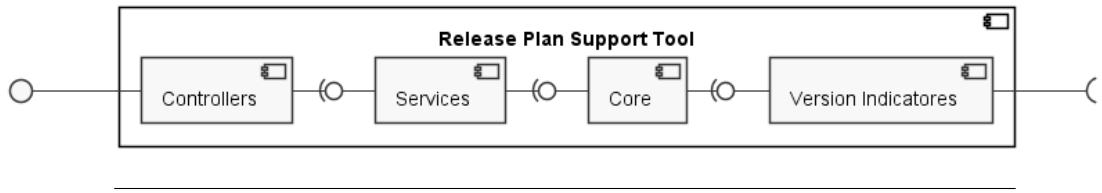

Figura 5.10: Diagrama de Componentes do Jira Integrator

Na Figura 5.10 realça-se o componente Version Indicators, responsável por realizar os cálculos relativos aos indicadores estatísticos de cada versão.

Sumário

Acima foi apresentado o desenho lógico do sistema em consideração nesta dissertação.

Como mencionado anteriormente, o principal foco no desenho deste sistema foi a segregação de responsabilidades em componentes distintos para que cada componente integrante da solução final tivesse as suas responsabilidades bem definidas.

No final, o principal objetivo foi desenhar um *software* fácil de manter e expandir.

5.2.3 Vista de Implementação

A vista de implementação visa oferecer uma perspetiva do projeto do ponto de vista do programador em relação ao sistema e à organização do seu código-fonte.

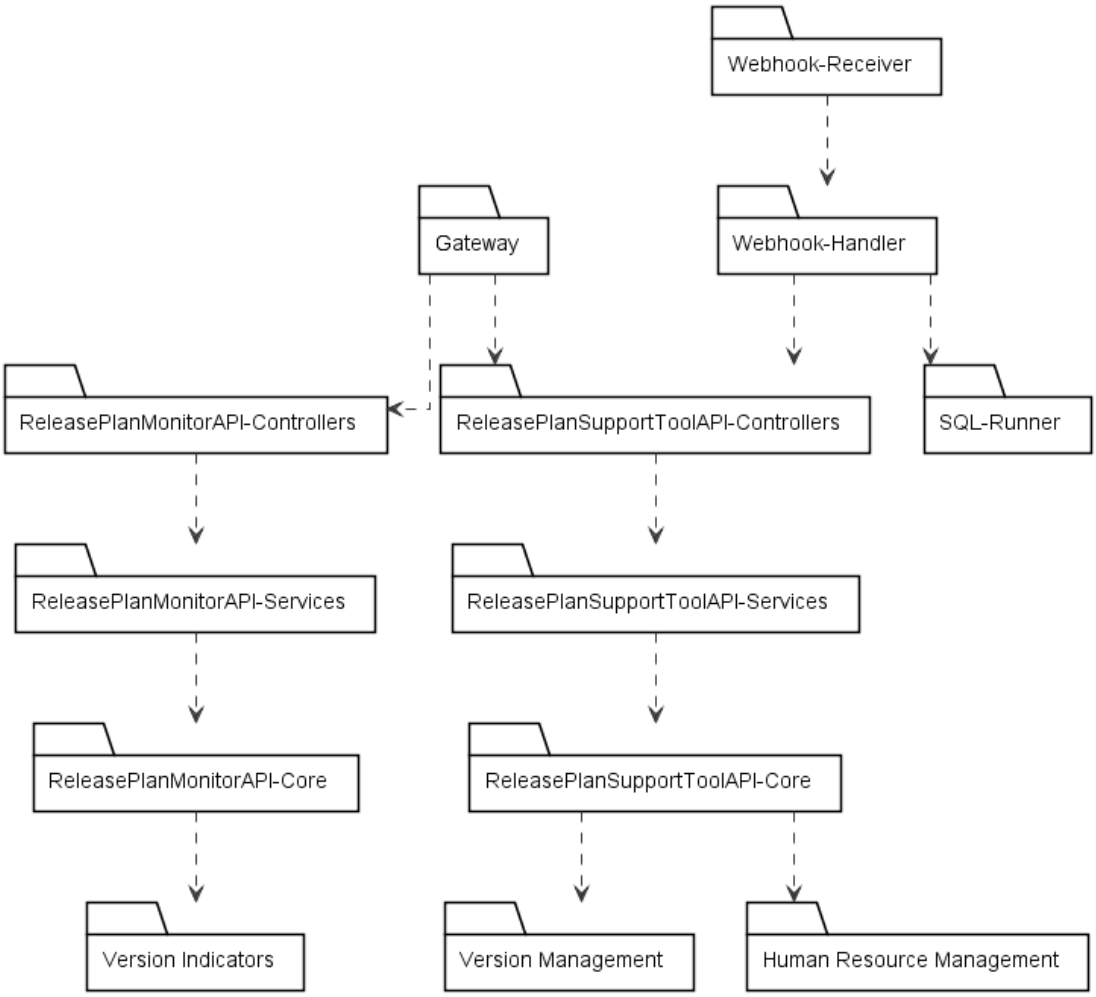

Figura 5.11: Diagrama de Packages do Sistema

Assim, através da análise à Figura 5.11, é possível observar que cada um dos componentes supra-mencionados foi dividido em diferentes *packages* de código-fonte.

Posto isto, e para melhor compreensão de todo o projeto, em seguida será apresentado um diagrama de implementação que serviu como base para estruturar cada implementação dos componentes (SQL-Runner, Jira-Integrator, Release Plan Support Tool e Release Monitor).

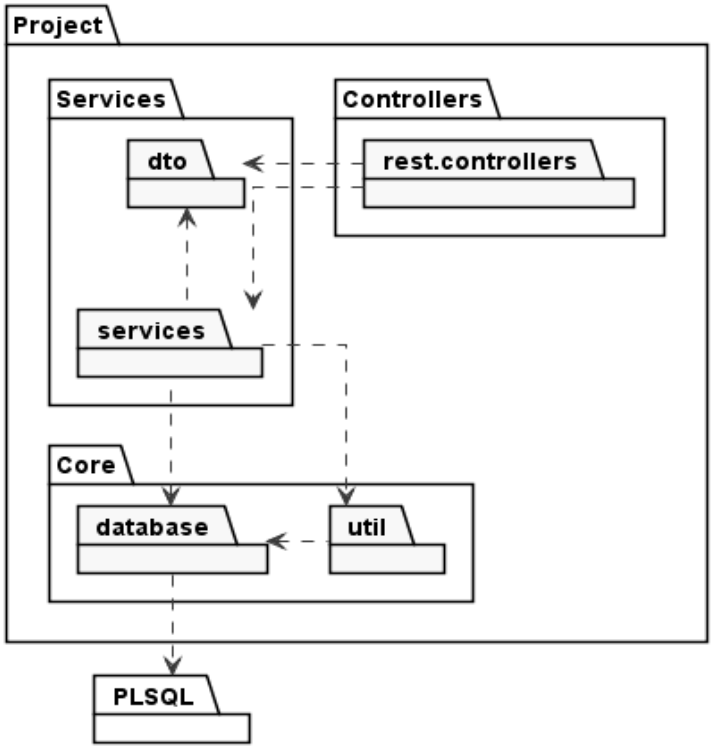

Figura 5.12: Diagrama de Packages Referência

Todos os projetos de código-fonte implementados seguiram uma lógica de implementação semelhante à apresentada no diagrama da Figura 5.12 com o objetivo de que cada um dos *packages* apenas tivesse uma responsabilidade única e não atribuir responsabilidades diversas ao mesmo *pacakge*.

5.2.4 Vista de Implantação

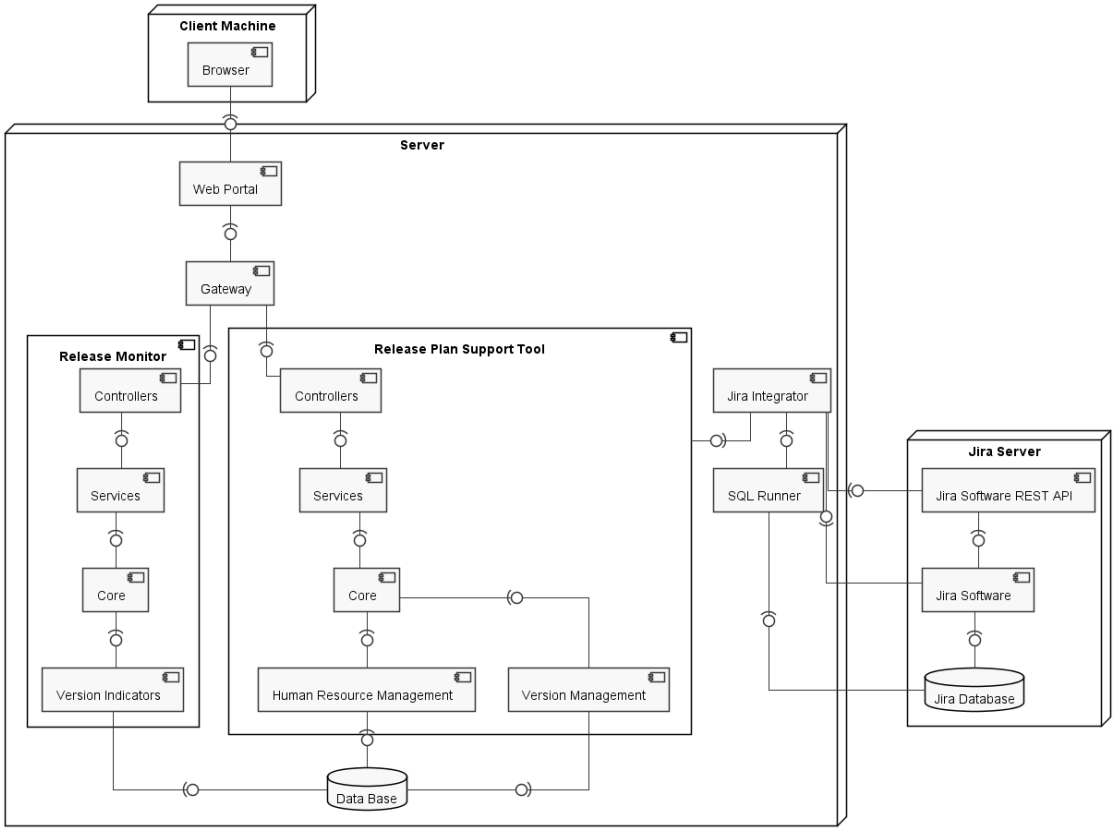

Figura 5.13: Diagrama de Implantação

Segundo o diagrama da Figura 5.13 é possível verificar apenas uma máquina servidor é utilizada, o Server, esta é um máquina Linux que aloja quer os componentes de *middleware* - Controllers; Services; Core - quer a base de dados - Data Base. Para além disto, nesta máquina está alojado, também, o Jira Integrator e o Web Portal.

Na imagem também se faz referência ao Jira Server da ALERT que se encontra alojado noutra máquina e à máquina cliente, onde, através de um *browser* os utilizadores podem interagir com a plataforma.

5.2.5 Vista de Processos

Nesta subsecção irão ser apresentados três processos que exemplificam a comunicação entre os componentes integrantes do projeto.

Criar uma versão

Uma versão é criada mediante um evento despoletado no Jira *Software* aquando a criação de um *issue*.

Na ALERT, há diversos *issue types* customizados. Neste contexto, um evento associado à criação de um *issue* com um *issue type "Version"* é despoletado e capturado pelo componente responsável por lidar com os eventos provenientes do Jira *Software*, o Jira Integrator.

Assim sendo, um evento chega ao Jira Integrator sob a forma de um *Webhook*, ou seja, um pedido POST [2] que chega ao Jira Integrator com uma mensagem no formato JSON, o evento.

Posto isto, o Jira Integrator utiliza o SQL Runner para executar uma consulta à base de dados do Jira *Software* para que todos os *issues* associados à versão que foi criada na plataforma Jira sejam carregados.

Terminada a consulta, é criado um JSON com todos os dados relativos à versão e enviada uma mensagem através de um pedido REST ao ReleasePlanSupportAPI para que seja persistida a nova versão na base de dados da ferramenta e uma data para o fim desta nova versão possa ser sugerida pela ferramenta.

A Figura 5.14 representa os processos mencionados anteriormente para a criação de uma versão na ferramenta.

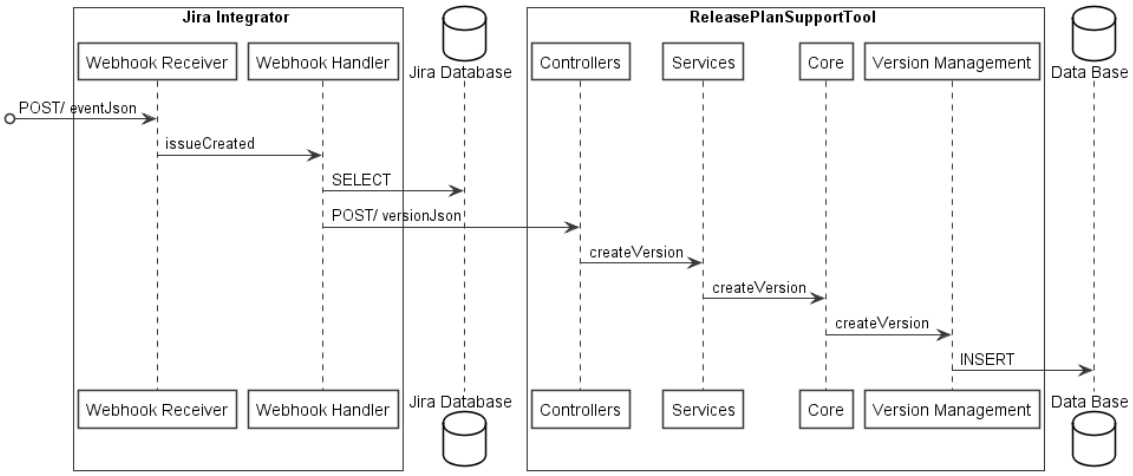

Figura 5.14: Diagrama de Sequência FR-06

Carregar dados dos Recursos Humanos de um ficheiro Excel

De acordo com a Figura 5.15, um membro da Equipa de *Release Planning* carrega um ficheiro Excel e, posto isso, um pedido HTTP é enviado para a ReleasePlanSupportAPI com o ficheiro. Após isto, o ficheiro é tratado e convertido para objetos Java, para que, após a conversão, todos os dados relativos aos recursos humanos sejam persistidos na base de dados.

[2] O método HTTP POST envia dados ao servidor.

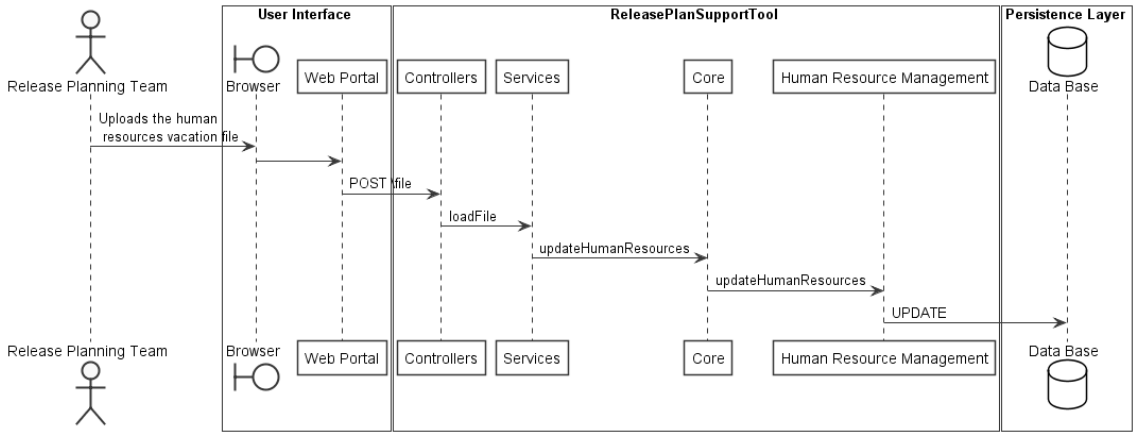

Figura 5.15: Diagrama de Sequência FR-01

Consultar os indicadores estatísticos de uma versão

Membro da Equipa de Desenvolvimento deseja consultar os indicadores estatísticos de uma versão.

Após navegarem na plataforma e selecionarem a versão da qual desejam consultar os dados estatísticos, um pedido HTTP é enviado à ReleaseMonitorAPI, esse pedido leva a que seja efectuada uma chamada a um conjunto de componentes PL/SQL guardados na base de dados, comummente denominados por *stored functions*, que devolvem os indicadores à ReleaseMonitorAPI. No final, é construído e devolvido à plataforma um *JSON* com todos os indicadores da versão.

Na Figura 5.16 podem está ilustrado o processo detalhado anteriormente.

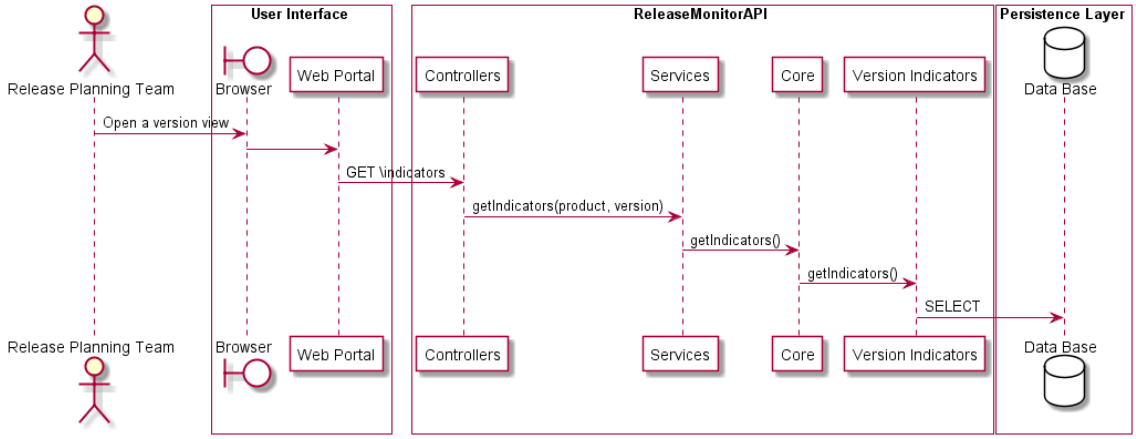

Figura 5.16: Diagrama de Sequência FR-09

5.2.6 Vista de Cenários

Por fim, a Vista de Cenários usualmente é capturada em Casos de Uso. Como referido no no Capitulo 4 todos os Casos de Uso podem ser consultados no Anexo B.

Capítulo 6

Implementação da Solução

Este capítulo tem como objetivo apresentar como foi realizada a implementação da solução tento em conta a arquitetura definida no Capítulo 5.

6.1 Contexto Tecnológico

Para ir de encontro com a *stack* tecnológica da ALERT, e, também, por requisito da Equipa de *Release Planning* todo o *software* foi desenvolvido da seguinte forma:

- Criação de uma *Single-Page Application (SPA)* com recurso a AngularJS, para oferecer uma interface gráfica com o utilizador.
- Criação de uma camada *middleware* responsável por expor funcionalidades em REST com recurso a Java.
- Criação de componentes PL/SQL para lidar com toda a lógica do problema.

Para além disto, foi utilizada uma *Framework* que a ALERT desenvolveu para que o desenvolvimento da comunicação entre os componentes Java e os componentes PL/SQL fosse mais rápido.

6.2 Autenticação e Autorização

Autenticação e Autorização são dois processos necessários para que se possa aceder a aplicações em com segurança.

Enquanto que, autenticação "é o processo de provar que é quem diz ser", autorização "é o ato de conceder permissão a uma parte autenticada para fazer algo"[54].

Dito isto, esta secção tem como objetivo apresentar como foi implementada a autenticação e a autorização para este projeto.

Dado que, a plataforma é para uso interno na ALERT a escolha passou por implementar um processo de autenticação com recurso às credenciais de utilizador existentes no *Active Directory (AD)* da ALERT através do *Lightweight Directory Access Protocol (LDAP)*. Desta forma, qualquer colaborador da ALERT consegue aceder à plataforma.

Sendo que toda a camada de *middleware* foi desenvolvida em Java com recurso à *framework* Spring, o processo de autenticação foi implementado através da utilização das bibliotecas

disponibilizadas pelo Spring Security[1]. Para além disso, a utilização do LDAP também foi implementada com recurso a bibliotecas do Spring, o Spring LDAP[2].

Desta forma, sempre que um pedido *Hypertext Transfer Protocol (HTTP)* é efectuado ao Gateway esse mesmo pedido deve conter no seu *header* um campo com o seguinte formato **Authorization: Basic <credentials>**, onde **<credentials>** é uma codificação *Base64*[3] do nome de utilizador e da senha unidos por dois pontos (**:**), ou seja, é necessário que seja efectuada uma Autenticação HTTP 'Básica', de acordo com especificado no *standard* RFC 7617 [55], para aceder aos recursos expostos pelo Gateway.

Assim, sempre que pedido HTTP é feito ao Gateway é, também, executada uma autenticação LDAP. O Excerto 6.1 apresenta um exemplo de uma possível configuração para que uma autenticação LDAP seja possível.

```
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter
    {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
    Exception {
        auth
                .ldapAuthentication()
                .userDnPatterns("uid={0},ou=people")
                .groupSearchBase("ou=groups")
                .contextSource()
                .url("ldap://localhost:8389/dc=springframework,dc=org")
                .and()
                .passwordCompare()
                .passwordEncoder(new BCryptPasswordEncoder())
                .passwordAttribute("userPassword");
    }
    ...
}
```

Listing 6.1: Exemplo de uma configuração LDAP em Java

Após a descodificação das credencias que foram transportadas no *header* do pedido, estas são comparadas com as credenciais existentes no AD. Caso a comparação tenha sucesso, é permitido o acesso aos recursos expostos pelo Gateway, caso tenha insucesso um código de resposta de emphstatus de erro - **401 Unauthorized** - é enviado ao cliente HTTP.

Dado que o Gateway é o único ponto de acesso para as duas outras APIs, foram configuradas opções de acesso para cada uma das APIs através da implementação do método "configure"da classe "WebSecurityConfigurerAdapter". O seguinte trecho de código 6.2 representa uma possível implementação.

[1] https://spring.io/projects/spring-security
[2] https://spring.io/projects/spring-ldap
[3] Método para codificação de dados transferidos na Internet

```
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter
    {

     ...

     @Override
     public void configure(HttpSecurity http) throws Exception {
         http
                 .authorizeRequests()
                 .antMatchers("/gateway/releaseplan/monitor/rest/**").
    access("hasRole('DEVELOPERS')")
                 .antMatchers("/gateway/releaseplan/tool/rest/**").access
    ("hasRole('MANAGERS')")
                 .anyRequest()
                 .fullyAuthenticated()
                 .and()
                 .httpBasic();
     }
}
```

Listing 6.2: Exemplo de uma configuração HttpSecurity em Java

Para o caso em apreço, apenas os membros da Equipa de *Release Planning* podem ter acesso aos recursos da ReplreasePlanningToolAPI pelo que foram aplicadas restrições de movo a garantir isto.

No final, através desta implementação, está garantido o acesso à plataforma a todos os colaboradores da ALERT com a devida autorização para o fazerem.

6.3 Base de Dados

A base de dados escolhida para a persistência dos dados foi a Oracle Database XE [56]. Esta escolha deveu-se ao facto desta base de dados ir de encontro com a *stack* tecnológica da ALERT, ser fácil de instalar e fácil de gerir.

6.3.1 Definição de Tabelas da Base de Dados

Por se tratar de uma base de dados Oracle, a definição das tabelas da base de dados foi com recurso a um subconjunto de instruções *Structured Query Language (SQL)*. Este subconjunto de instruções SQL, muitas vezes denominado como *Data Definition Language (DDL)*, é um conjunto de comandos *SQL* que definem a estrutura da base de dados, incluindo linhas, colunas, tabelas, índices, entre outros objetos.

O Excerto 6.3 é uma declaração **CREATE** que define uma tabela das necessárias para o projeto, e as respetivas colunas.

```
CREATE TABLE HUMAN_RESOURCE (
  ID_HUMAN_RESOURCE NUMBER(24) NOT NULL,
  ID_TEAM NUMBER (24) NOT NULL,
  MECAN_NUMBER NUMBER(6) NOT NULL,
  EXPERIENCE_LVL NUMBER(1) NOT NULL,
  NAME CHAR(50) NOT NULL,
  LAYER CHAR(10) NOT NULL
  );
```

Listing 6.3: Exemplo da definição da tabela HumanResource com o comando *SQL* para criação

O comando **CREATE TABLE** permite, então, a criação de uma tabela e a definição de suas colunas na base de dados. A definição de cada coluna tem, obrigatoriamente, um nome para a coluna e o seu tipo de dados. Para além disso, ao definir uma coluna é possível declarar se esta aceita valores *NULL*, ou não, definir se se trata de uma chave-primária e definir restrições para os seus valores.

Para além da declaração **CREATE** outras declarações existem. Sendo estas:

- **DROP** - utilizada para remover tabelas, índices ou outros objetos da base de dados;
- **ALTER** - utilizada para modificar um objeto já existente na base de dados;
- **TRUNCATE** - apaga imediatamente todo o conteúdo de uma tabela ou outro objeto que contenha dados.

Existem ainda, declarações de integridade referencial. Estas declaração existem para definir as relação entre as tabelas de uma base de dados relacional e são as seguintes: **PRIMARY KEY**, **UNIQUE** e **FOREIGN KEY**.

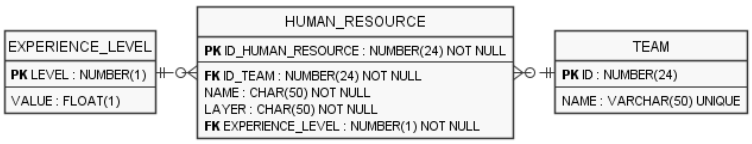

Figura 6.1: Excerto do Modelo Relacional de Base de Dados

A Figura 6.1 apresenta um excerto do Modelo de Dados do projeto. Pela sua análise é possível verificar que existem relações existentes entre as tabelas. Assim sendo, há a necessidade de definir estas relações recorrendo às declarações de integridade referencial. Neste caso em específico, todas as relações foram criadas após a criação da tabela através do comando *SQL* **ALTER TABLE**, invés de serem definidas em conjunto com a definição das colunas, como é possível observar na Listagem 6.4.

```
ALTER TABLE HUMAN_RESOURCE ADD (
 CONSTRAINT HUMAN_RESOURCE_PK PRIMARY KEY (ID_HUMAN_RESOURCE)
 );
```

Listing 6.4: Exemplo da definição da chave-primária ID Human Resource com o comando *SQL* para alteração

Para além de tudo o que foi mencionado, é possível com comandos SQL restringir os valores que uma determinada coluna pode guardar. Por exemplo, a coluna "LAYER"da tabela

"HUMAN RESOURCE"apenas pode guardar um de três valores: "UX", "MW"ou "DB". Para garantir que esta restrição é cumprida foi executado o seguinte comando,

```
ALTER TABLE HUMAN_RESOURCE
 ADD CONSTRAINT HUMAN_RESOURCE_LAYER_CHK CHECK (LAYER IN ('UX', 'MW', '
    DB')) ;
```

Listing 6.5: Exemplo de uma restrição aplicada a uma coluna

Sumarizando, ao longo desta subsecção foram apresentadas evidências de como foi construído o modelo de dados por de traz deste projeto. Para além disto, foi demonstrada a capacidade da linguagem SQL para definir objectos em bases de dados Oracle.

6.3.2 Definição de Packages PL/SQL

Procedural Language/Structured Query Language (PL/SQL) é uma linguagem de programação estruturada que estende a linguagem SQL. Esta linguagem permite que manipulação de dados seja incluída em unidades de programas. Dito isto, *Packages* PL/SQL são objetos que agregam tipos, variáveis e subprogramas PL/SQL que estão logicamente relacionados entre si.

Como já mencionado, a implementação da lógica e do domínio do problema em questão foi realizada em *Packages PL/SQL*. Assim sendo, em seguida irá ser ilustrado o processo de implementação desses *packages*.

Cada *package* PL/SQL tem duas partes obrigatórias:

- Especificação do *package*;
- Corpo do *package*.

A especificação do *package* é a interface do *package* e, como tal, apenas tem declarações de tipos, variáveis, métodos, etc... Por sua vez, o corpo do *package* contém o código dos diversos métodos declarados na especificação.

A criação de *packages* PL/SQL, assim como os demais objetos das bases de dados Oracle, é através do comando **CREATE**. O Excerto 6.6 apresenta como pode ser criada a especificação de um *package* e o Excerto **??** apresenta como pode ser criado um corpo de um *package*.

```
CREATE OR REPLACE PACKAGE PK_PRODUCT_MNG IS
    ...
END PK_PRODUCT_MNG;
```

Listing 6.6: Exemplo da criação da especificação de um *package* PL/SQL

Com o intuito de organizar o código e atribuir responsabilidades únicas a cada um dos objetos PL/SQL foram criados quatro *packages*. A Figura 6.2 apresenta um diagrama que ilustra os *packages* que compõe a solução.

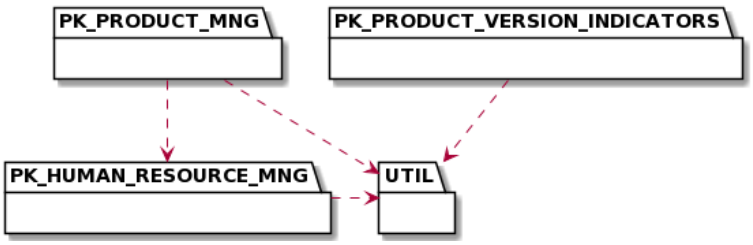

Figura 6.2: Diagrama de Packages PL/SQL

Cada um destes *packages* contém a definição de vários *PROCEDURES* e/ou *FUNCTIONS* que, basicamente, são subprogramas que executam uma determinada tarefa. A combinação de todas as unidades de código resultam em programas maiores, pelo facto de cada um destes subprogramas poder ser chamado por outro subprograma e poder chamar outros subprogramas.

No Excerto 6.7 está ilustrado um exemplo de uma *FUNCTION*. Esta *FUNCTION* tem como tarefa alterar o nivel de experiência de um determinado recurso humano.

```
1  FUNCTION CHANGE_HR_EXP_LVL(I_HR_ID    IN HUMAN_RESOURCE.ID_HUMAN_RESOURCE
       %TYPE,
2                                 I_EXP_LVL IN EXPERIENCE.LVL%TYPE) RETURN
       BOOLEAN IS
3    BEGIN
4      UPDATE HUMAN_RESOURCE HR
5         SET HR.EXP_LVL = I_EXP_LVL
6       WHERE HR.ID_HUMAN_RESOURCE = I_HR_ID;
7
8      IF (SQL%ROWCOUNT = 0) THEN
9        RAISE_APPLICATION_ERROR(-20001,
10                               'No such record with id ' || I_HR_ID ||
11                               ' exists');
12     END IF;
13     RETURN TRUE;
14   END CHANGE_HR_EXP_LVL;
```

Listing 6.7: Exemplo da criação da especificação de um *package* PL/SQL

Concluindo, a implementação da lógica da solução em PL/SQL, para além de ir de encontro com o praticado na ALERT, permite que seja realizada a separação das funcionalidades do programa em módulos independentes.

6.4 Middleware

Com o intuito de expor, através de uma REST API, as funcionalidades desenvolvidas em PL/SQL, foi desenvolvida, em JAVA, com recurso à *framework* Spring, uma camada de *middleware*. Assim sendo, aqui serão expostos alguns detalhes da implementação dessa camada de *middleware*.

Em primeiro lugar, para que seja possível expor as funcionalidades desenvolvidas em PL/SQL é necessário que seja possível invocar os *packages* que as contém. Essa implementação foi através da utilização de um projeto da ALERT, o "Merlin-InvocationContext", que oferece

mecanismos que facilitam a implementação necessária para invocar quaisquer *packages/procedures/functions* guardados numa base de dados.

```
1 public class ClearHrVacationsDBInvocationContext extends
      GenericDBInvocationContext implements IDBInvocationContext {
2     ...
3 }
```

Listing 6.8: Exemplo de uma Classe Java que invoca Objetos PL/SQL

Implementada a comunicação entre a aplicação Java e a base de dados, segue-se a implementação de um *endpoint* REST capaz de expor a funcionalidade.

```
1  @RestController
2  @RequestMapping("/humanresources")
3  public class HumanResourceController {
4
5    @Autowired
6    IHumanResourceService humanResourceService;
7
8    @GetMapping(value = "/{idHumanResource}", produces = "application/json
       ")
9    public ResponseEntity<ResponseDto<HumanResourceDto>> getHumanResource(
       @PathVariable BigDecimal idHumanResource)
10        throws Exception {
11      return ResponseEntityBuilder.buildResponse(humanResourceService.
       getHumanResource(idHumanResource));
12   }
13 }
```

Listing 6.9: Exemplo de um serviço REST

O Extrato 6.9 representa uma implementação de um serviço REST. Como já mencionado, os projetos de *middleware* foram implementados com recurso à *framework* Spring, por isso, no exemplo é possível identificar algumas anotações Java, como por exemplo **@RestController**; **@RequestMapping("/humanresources")**; **@GetMapping(value = "/idHumanResource", produces = "application/json")**, que são da *framework* e alteram o comportamento da classe para que esta se comporte como um serviço *REST*. No exemplo, foi criado um método que realiza uma consulta à base de dados.

No final, esta camada de *middleware* é uma API REST e como tal expões serviços REST. O endereço destes serviços segue o seguinte formato:

- **HTTP://[IP]:[PORT]/releaseplan/tool/rest**, para os serviços da ReleasePlanSupportToolAPI;
- **HTTP://[IP]:[PORT]/releaseplan/moniotr/rest**, para os serviços da RelasePlanMonitorAPI.

6.5 Integração com o Jira Software

Para que a plataforma atinja o seu propósito foi necessário integra-la com o Jira *Software*. A integração com o Jira *Software* passou pela criação de um projeto capaz de receber e tratar os eventos publicados pelo Jira. Ou seja, desenvolver um projeto que exponha *endpoints* para que o Jira os consuma.

A primeira fase da implementação passou pela criação de serviços REST capazes de receber os eventos do Jira. Por se tratar de um projeto Java com Spring, esta implementação é muito semelhante ao exemplificado na Secção 6.4.

O segundo desafio da implementação, surge com a necessidade de desserializar os evento emitidos pelo Jira que chegam à API numa mensagem JSON, para isso, foi necessário a criação de um objeto Java com a função de Data Transfer Object (DTO). Com este objeto (Extrato 6.10), é possível converter a mensagem JSON num objeto Java para que os dados da mensagem possam ser manipulados pela aplicação Java. Esta conversão é realizada pela *framework* Spring e tem por base uma biblioteca de processamento de dados - Jackson.

```
1  @Data
2  public class JiraEventDto implements DTO {
3    private static final long serialVersionUID = 3079651269138706512L;
4
5    private String webhookEvent;
6    private JiraIssueDto issue;
7    private Date timestamp;
8    private ChangeLogDto changelog;
9
10   @Override
11   public String toString() {
12     return "JiraEventDto [webhookEvent=" + webhookEvent + ", issue=" +
       issue + ", timestamp=" + timestamp
13         + ", changelog=" + changelog + "]";
14   }
15 }
```

Listing 6.10: Exemplo de um DTO

Após a desserialização da mensagem temos o evento criado pelo Jira num objeto Java e é necessário tratar o evento. Para isso foram implementadas uma série de classes, ilustradas na Figura 6.3, responsáveis por tratarem dos eventos recebidos.

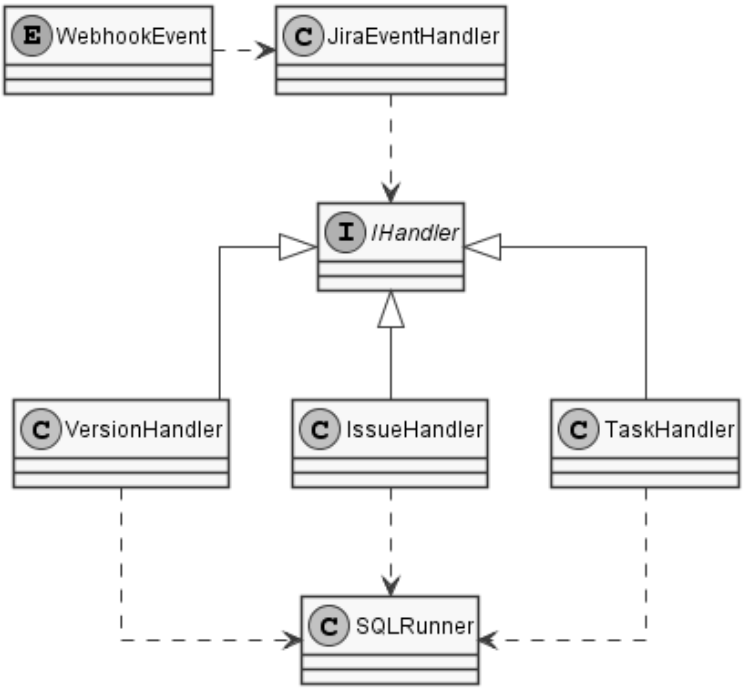

Figura 6.3: Diagrama de classes para tratar um evento do Jira

Assim sendo, o Jira Integrator é uma aplicação Java que recebe eventos sob a forma de mensagem JSON e converte-os para objetos. Após isso, mediante o tipo de evento diferentes consultas são realizadas à base de dados do Jira. Por exemplo, se o evento for "*jira:issue created*"e esse *issue* for do tipo *"Version"* será realizada uma consulta à base de dados do Jira para ir buscar todos os *issues* daquela versão. Após a consulta ser realizada, é criado uma JSON como todos os dados e enviada uma mensagem para o ReleasePlanSupportTool.

6.6 Cálculo dos Indicadores Estatísticos

Um dos objetivos desta plataforma é fornecer indicadores estatísticos sobre as versões para que se possa acompanhar o progresso da mesma. Assim, nesta secção irão ser apresentados a, titulo de exemplo, algumas implementações em PL/SQL relativas aos indicadores. A lista completa de indicadores com sua explicação e implementação pode ser consultada no Anexo C.

6.6.1 Total macro issues planned

Para obter este indicador é necessário somar todas as estimativas de todos os *issues* da versão em questão. Este indicador é obtido da seguinte forma:

```
FUNCTION TOTAL_MACRO(I_PRODUCT          IN VERSION.PRODUCT%TYPE,
                     I_PRODUCT_VERSION  IN VERSION.PRODUCT_VERSION%TYPE)
    RETURN NUMBER IS
    O_TOTAL_MACRO   NUMBER
  BEGIN
    SELECT SUM(ESTIMATION)
      into O_TOTAL_MACRO
      FROM ISSUE
     WHERE ISSUE.PRODUCT = I_PRODUCT
       and ISSUE.PRODUCT_VERSION = I_PRODUCT_VERSION;
    RETURN O_TOTAL_MACRO;
  END TOTAL_MACRO;
```

Listing 6.11: Implementação do total macro issues planned

6.6.2 Total macro issues planned by team and by layer

Este indicador é bastante semelhante ao anterior, o objetivo passa por calcular o total das estimativas de cada camada, de cada equipa, para que seja fácil visualizar a distribuição do esforço necessário para a *release* pelas diversas equipas de desenvolvimento. O Excerto C.3 apresenta a implementação deste indicador.

```
FUNCTION TOTAL_TEAM_MACRO_PER_LAYER(I_TEAM              IN TEAM.NAME%TYPE,
                                    I_PRODUCT           IN VERSION.
    PRODUCT%TYPE,
                                    I_PRODUCT_VERSION IN VERSION.
    PRODUCT_VERSION%TYPE,
                                    O_MACRO_PER_LAYER OUT
    T_LAYER_MACRO)
     RETURN BOOLEAN IS
  BEGIN
    OPEN O_MACRO_PER_LAYER FOR
      SELECT TASK.LAYER, SUM(TASK.ESTIMATION) AS MACRO
      FROM TEAM, ISSUE, COMPONENT, TASK
      WHERE ISSUE.PRODUCT = I_PRODUCT
      AND ISSUE.PRODUCT_VERSION = I_PRODUCT_VERSION
      AND TASK.PARENT_KEY = ISSUE.ISSUE_KEY
      AND COMPONENT.NAME = TASK.COMPONENT
      AND TEAM.ID_TEAM = COMPONENT.ID_TEAM
      AND TEAM.NAME = I_TEAM
      GROUP BY TASK.LAYER;
    RETURN TRUE;
  END TOTAL_TEAM_MACRO_PER_LAYER;
```

Listing 6.12: Implementação do total macro issues planned by team and by layer

6.6.3 Total development days

O "Total development days"é obtido através da subtracção de duas datas. Cada versão tem uma data de "Deadline Planning", que marca o ínicio do desenvolvimento da versão, e uma data de "Deadline Development", que marca o seu fim. Assim, para obter os dias de desenvolvimento basta apenas subtrair "Deadline Development"a "Deadline Planning".

```
FUNCTION BUSINESS_DAYS_INTERVAL(BEGIN_TIMESTAMP TIMESTAMP,
                                END_TIMESTAMP   TIMESTAMP)
    RETURN NUMBER IS
    DAYS_IN_BETWEEN      NUMBER := 0;
    BUS_DAYS_IN_BETWEEN  NUMBER := 0;
    BEGIN_DATE           DATE := CAST(BEGIN_TIMESTAMP AS DATE);
    END_DATE             DATE := CAST(END_TIMESTAMP AS DATE);
  BEGIN
    WITH DAYS AS
     (SELECT BEGIN_DATE + SEQ AS DAY_DATE,
             TO_CHAR(BEGIN_DATE + SEQ, 'D') DAY_OF_WEEK
        FROM (SELECT ROWNUM - 1 SEQ
                FROM (SELECT 1
                        FROM DUAL
                      CONNECT BY LEVEL <= (END_DATE - BEGIN_DATE) + 1))
       ORDER BY 1)
    SELECT END_DATE - BEGIN_DATE AS DAYS_INBETWEEN,
           COUNT(1) BUSINESS_DAYS_INBETWEEN
      INTO DAYS_IN_BETWEEN, BUS_DAYS_IN_BETWEEN
      FROM DAYS
     WHERE DAYS.DAY_OF_WEEK NOT IN (7, 1);

    RETURN BUS_DAYS_IN_BETWEEN;
  END BUSINESS_DAYS_INTERVAL;
```

Listing 6.13: Implementação de uma função para calcular os dias de trabalho entre duas datas

A Função 6.13 recebe duas datas e retorna o número de dias úteis que existe entre elas, esta função utilizada para obter a diferença entre "Deadline Development"a "Deadline Planning"o que resulta no indicador "Total development days".

6.6.4 Total remaining macro issues planned

Este indicador resulta da soma de todas as estimativas de todos os *issues* cujo o seu progresso é inferior a 100%, ou seja, todos os issues que não estão totalmente implementados, como pode ser observado na Listagem C.5

```
FUNCTION TOTAL_REMAINING_MACRO(I_PRODUCT          IN VERSION.PRODUCT%TYPE
    ,
                               I_PRODUCT_VERSION IN VERSION.
    PRODUCT_VERSION%TYPE)
    RETURN ISSUE.ESTIMATION%TYPE IS
    O_TOTAL_MACRO ISSUE.ESTIMATION%TYPE;
  BEGIN
    SELECT SUM(ESTIMATION)
      INTO O_TOTAL_MACRO
      FROM ISSUE
     WHERE ISSUE.PROGRESS < 1.0
       AND ISSUE.PRODUCT = I_PRODUCT
       AND ISSUE.PRODUCT_VERSION = I_PRODUCT_VERSION;
    RETURN O_TOTAL_MACRO;
  END TOTAL_REMAINING_MACRO;
```

Listing 6.14: Implementação do total remaining macro issues planned

6.6.5 Days progress

A diferença entre o "Total development days"e o "Remaining Development Days"a dividir pelo "Total development days"resultará num valor decimal que representa o progresso dos dias do desenvolvimento da *release*.

```
1  FUNCTION DAYS_PROGRESS(I_PRODUCT            IN VERSION.PRODUCT%TYPE,
2                         I_PRODUCT_VERSION IN VERSION.PRODUCT_VERSION%TYPE
       )
3    RETURN NUMBER IS
4    O_TOTAL_DEV_DAYS NUMBER;
5    O_REMAINING_DEV_DAYS NUMBER;
6  BEGIN
7   O_TOTAL_DEV_DAYS := TOTAL_DEV_DAYS(I_PRODUCT => I_PRODUCT,
8                                      I_PRODUCT_VERSION =>
       I_PRODUCT_VERSION);
9   O_REMAINING_DEV_DAYS := O_REMAINING_DEV_DAYS(I_PRODUCT => I_PRODUCT,
10                         I_PRODUCT_VERSION => I_PRODUCT_VERSION);
11  RETURN ((O_TOTAL_DEV_DAYS - O_REMAINING_DEV_DAYS)/O_TOTAL_DEV_DAYS)*
      100;
12 END DAYS_PROGRESS;
```

Listing 6.15: Implementação do days progress

6.7 Cálculo da Data Prevista

O cálculo de uma data prevista para o fim do desenvolvimento de uma *release* é o objetivo fulcral deste projeto. Toda a plataforma foi pensada e construída para que fosse possível à equipa de *release planning* obter esta data com facilidade e rapidez.

Este cálculo é despoletado sempre que ocorre um evento na aplicação, por exemplo, um novo ficheiro de férias é carregado, ou então um novo *issue* é adicionado à *version*.

Deste modo, uma vez mais, recorreu-se às possibilidades que a *framework* Spring oferece para implementar um solução de acordo com o problema.

Sempre que acontece uma alteração na plataforma é registado um evento no contexto da aplicação (Listagem 6.16), isto ocorre através da interface ApplicationEventPublisher que é uma interface que encapsula a funcionalidade de publicação de eventos oferecida pela *framework*.

```
@Service
public class VersionServiceImpl implements IVersionService {

   ...

   @Autowired
   private ApplicationEventPublisher publisher;

     @Override
   public void defineHumanResourceAlocation(String product, String
     version, BigDecimal hrId, Double alocation)
       throws InvocationException, SQLException {
     ...
     publisher.publishEvent(new CostumEvent(this, product, version));
   }
}
```

Listing 6.16: Exemplo da publicação de um evento com ApplicationEventPublisher

Agora, com o evento registado no contexto da aplicação, é necessário escutar o evento e manipula-lo. Assim, através da implementação de um "Event Listener"é possível receber o evento, interpreta-lo e agir em conformidade. Neste caso em específico, qualquer evento que chegue ao seu *listener* deverá desencadear o cálculo para obter a data prevista para o fim do desenvolvimento da *release*.

6.7.1 Obtenção da Data

Finalizada a explicação de como é despoletado o calculo, segue a explicação do calculo em si.

Em primeiro lugar, é necessário determinar, para todas as equipas envolvidas no desenvolvimento da *release*, qual é a sua capacidade. Nesta caso, entenda-se capacidade o número de horas que a equipa tem disponíveis para trabalhar durante os dias previstos para o desenvolvimento da *release*.

Assim,

$$capacidade = (diasdetrabalhonecessario - ausencias) * horasdetrabalho \quad (6.1)$$

Calculadas as horas que a equipa tem disponíveis para a *release*, é necessário perceber quantas dessas horas a equipa irá "gastar"no desenvolvimento da *release*. Para obter a quantidade de horas que a equipa vai dispensar para a *release* basta multiplicar a capacidade pela percentagem de alocação da equipa à *release*.

No final, apenas é necessário perceber qual é a duração máxima estimada para a *release*. Como se considera que cada equipa trabalha em paralelo basta apenas obter a duração máxima estimada para cada equipa (Fórmula 6.2) e verificar qual é a que tem duração maior. Somando esta duração à data inicial do desenvolvimento da *release* chega-se à data final prevista para o desenvolvimento da *release*.

$$duracaoestimada = diasestimados/(capacidadedaequipa * alocacaodaequipa) \quad (6.2)$$

6.8 Testes

Para garantir o correto funcionamento dos métodos e funções da aplicação foram realizados 2 tipos de testes de *software*: Testes Unitários e Testes de Integração. Nesta capitulo serão apresentados os resultados desses testes e alguns exemplos ilustrativos.

6.8.1 Testes Unitários

teste de unidade é um método de teste de software pelo qual unidades individuais de código-fonte - conjuntos de um ou mais módulos de programa de computador, juntamente com dados de controle, procedimentos de uso e procedimentos operacionais associados - são testados para determinar se eles são adequados para uso

Testes Unitários são testes de *software* através dos quais unidades individuais de código-fonte são testadas para determinar se eles são adequados para serem utilizados [57].

Para esta plataforma foram realizados, na sua maioria, testes unitários a classes utilitárias de modo a garantir que as mesmas tinham o comportamento esperado. Para destes, foi realizado um teste unitário de modo a garantir que o processamento do ficheiro Excel estava devidamente implementado.

Sendo que os testes unitários foram implementados em Java, recorreu-se à *framework* JUnit 5 [58].

```
1  public class DateUtilTest {
2
3    @Test
4    @DisplayName("Should be possible to convert Date into LocalDateTime")
5    public void testConvertDateToLocalDate() throws ParseException {
6      Date date = new SimpleDateFormat("yyyy-MM-dd").parse("2020-7-12");
7
8      LocalDateTime result = DateUtil.convertDateToLocalDateTime(date);
9      LocalDateTime expected = LocalDateTime.of(2020, 7, 12, 0, 0, 0, 0);
10
11     assertEquals(result, expected, "Should be possible to convert Date
       into LocalDateTime");
12   }
13
```

Listing 6.17: Exemplo de um teste de unitário em Java

O Excerto 6.17 é um exemplo de um teste unitário.

6.8.2 Testes de Integração

Testes de Integração surgem após os testes unitários e é a fase de testes de *software* onde se testa o comportamento dos diferentes módulos combinados entre si. Testes de Integração são realizados para avaliar a conformidade de um sistema com requisitos funcionais especificados [59].

Os testes de integração a esta plataforma foram desenvolvidos com recurso à funcionalidades de testes de integração fornecidas pela plataforma Postman [60].

Esta plataforma oferece um desenvolvimento bastante simples de testes de integração sendo apenas ter algum conhecimento de Javascript. O objetivo destes testes era cobrir os métodos da camada *middleware* que fazem chamadas à base de dados.

O Excerto 6.18 é um exemplo ilustrativo de um teste de integração no Postman.

```
pm.test("Request should pass", function() {
    pm.response.to.have.status(200);
})

var data = [{
        "date": "2021-05-04",
        "halfDay": "N"
    },
    {
        "date": "2021-05-05",
        "halfDay": "N"
    },
    {
        "date": "2021-05-06",
        "halfDay": "N"
    },
    {
        "date": "2021-05-07",
        "halfDay": "N"
    },
    {
        "date": "2021-05-10",
        "halfDay": "N"
    },
    {
        "date": "2021-05-11",
        "halfDay": "N"
    },
    {
        "date": "2021-05-12",
        "halfDay": "N"
    },
    {
        "date": "2021-05-13",
        "halfDay": "N"
    },
    {
        "date": "2021-05-14",
        "halfDay": "N"
    },
    {
        "date": "2021-05-18",
        "halfDay": "N"
    },
    {
        "date": "2021-05-19",
        "halfDay": "N"
    },
    {
        "date": "2021-05-20",
        "halfDay": "N"
    },
    {
        "date": "2021-05-21",
        "halfDay": "N"
    }
];

```

```
pm.test("Response must be success", function() {
    pm.response.success === true;
})

pm.test("Response should be equal to object data", function() {
    pm.response.data === data;
})
```

Listing 6.18: Exemplo de um teste de integração no Postman

Foram executados testes para todos os *endpoints* disponibilizados pela camada de *middleware* e todos passaram como se pode confirmar na Figura 6.4.

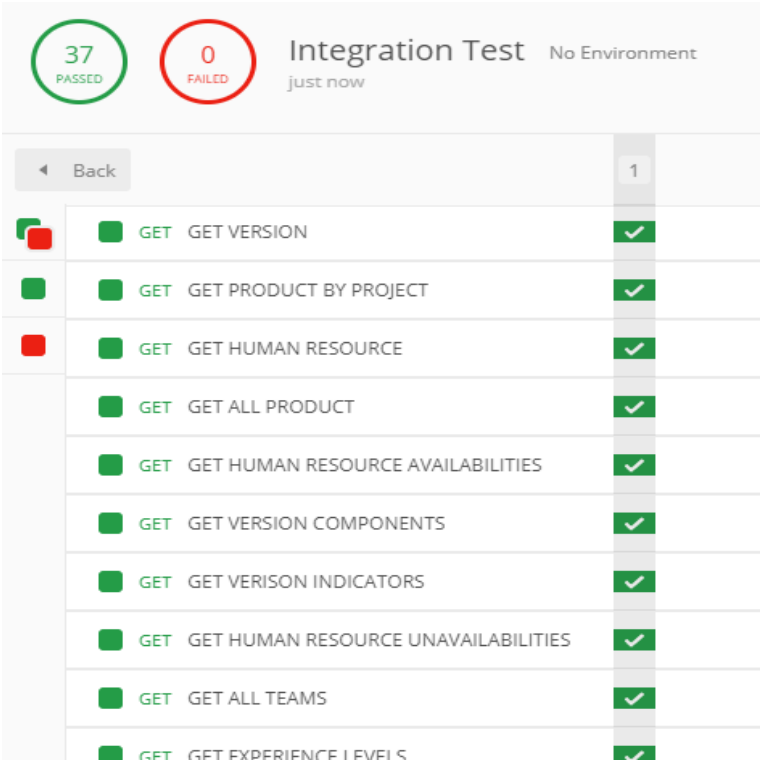

Figura 6.4: Resultado dos Testes de Integração

Capítulo 7

Avaliação e Experimentação

Este capítulo apresenta o processo de avaliação e experimentação do projeto desenvolvido e os seus resultados.

Primeiramente, serão apresentadas as métricas definidas para avaliar a solução. Em seguida, estão descritas as hipóteses formuladas. No final do capítulo, é apresentada a metodologia utilizada e os resultados da avaliação.

7.1 Métricas

Para uma avaliação global da solução foram consideradas três métricas:

- **Metrica 1:** Satisfação do utilizador quanto à utilização da plataforma;
- **Metrica 2:** Satisfação do utilizador quanto à performance do cálculo automático para a estimativa da data final do desenvolvimento de uma *release*;
- **Metrica 3:** Tempo total de execução e resposta.

7.2 Hipóteses

Posteriormente à definição das métricas, seguiu-se a formulação de hipóteses. Assim, por cada métrica foram formuladas 2 hipóteses.

7.2.1 Métrica 1:

- **Hipótese 1:** A usabilidade da plataforma tem um nível de satisfação superior a 3 numa escala de 0-5.
- **Hipótese 2:** A usabilidade da plataforma tem um nível de satisfação igual ou inferior a 3 numa escala de 0-5.

7.2.2 Métrica 2:

- **Hipótese 1:** A performance do cálculo tem um nível de satisfação superior a 3 numa escala de 0-5.
- **Hipótese 2:** A performance do cálculo tem um nível de satisfação igual ou inferior a 3 numa escala de 0-5.

7.2.3 Métrica 3:

- **Hipótese 1:** O sistema demora menos 1 segundo a responder.
- **Hipótese 2:** O sistema demora mais de 1 segundo a responder.

7.3 Metodologia

Definidas as métricas e as suas hipóteses é altura de avalia-las. Para desempenhar essa avaliação duas metodologias distintas serão utilizadas. Para avaliar a **Métrica 1** e **Métrica 2** foram realizados inquéritos de satisfação. Estes inquéritos foram disponibilizados à equipa de *release planning*. Para além de serem os principais utilizadores da plataforma, só eles conseguem avaliar o impacto da solução no desempenho das suas funções. No que diz respeito a última métrica, **Métrica 3**, a metodologia adotada foi a realização de testes de integração.

7.4 Análise de Resultados

Aqui é realizada uma análise dos resultados obtidos para cada uma das métricas anteriormente definidas. Além disso, é aferido se a plataforma cumpre com os casos definidos nas hipóteses da Secção 7.2.

7.4.1 Inquéritos de satisfação do utilizador

De modo a aferir se a plataforma corresponde ao que foi pedido pela equipa de *release planning* da ALERT, foi realizada uma apresentação da plataforma e das suas funcionalidades à equipa e depois foi pedido que esta respondesse a um pequeno questionário. Preferencialmente, devia ser realizada uma apresentação a todas a equipas de desenvolvimento pois certas funcionalidades serão do interesse geral das equipas, porém, entre o final de estágio e a escrita do presente documento, não foi possível realizar tal apresentação. Deste modo, a amostra é reduzida a três elementos, contudo, estes são os maiores interessados na plataforma e os responsáveis pela mesma.

Estes questionários apresentavam uma escala de satisfação entre 0 e 5, onde 0 é **INSATISFATÓRIO** e 5 é **BASTANTE SATISFATÓRIO**.

Feedback da usabilidade da plataforma

O questionário composto por 7 asserções apresentadas na tabela com o valor percentual de respostas para cada calor da escala anteriormente definida.

Tabela 7.1: Tabela de Requisitos Funcionais

Asserções	**0**	**1**	**2**	**3**	**4**	**5**
Design	0%	0%	33%	66%	0%	0%
Velocidade	0%	0%	0%	0%	100%	0%
Nível de usabilidade	0%	0%	0%	100%	0%	0%
Processo de alocação de recursos	0%	0%	0%	33%	33%	33%
Processo de consulta de indicadores estatísticos	0%	0%	0%	100%	0%	0%
Processo de carregamento de ficheiros	0%	0%	0%	66%	33%	0%
Integração com o Jira	0%	0%	0%	0%	33%	66%

Tendo em consideração as hipóteses formuladas para esta Métrica e os resultados do inquérito, onde a média das respostas obtidas é de aproximadamente 4, é possível afirmar que há satisfação geral na plataforma. Apesar disso, há certos aspetos a melhorar como o *design* e o nível de usabilidade.

Feedback da performance do cálculo

O questionário composto por 2 asserções apresentadas na tabela com o valor percentual de respostas para cada calor da escala anteriormente definida.

Tabela 7.2: Tabela de Requisitos Funcionais

Asserções	**0**	**1**	**2**	**3**	**4**	**5**
Desempenho do calculo da estimativa	0%	0%	0%	0%	66%	33%
Exatidão do calculo da estimativa	0%	0%	0%	0%	100%	0%

No que diz respeito à performance do cálculo, a média das respostas obtidas é de aproximadamente 4. Comparando este valor com o da hipótese, este é maior, por isto, é possível afirmar que o objetivo principal da plataforma foi cumprido com sucesso.

Tempo de resposta

Para avaliar os tempos de resposta, os testes de integração descritos na Secção 6.8.2, foram executados 10 vezes numa máquina com um processador Intel Core Duo CPU @ 2.93GHz, com a base de dados alojada num servidor interno e com ligação à rede interna da ALERT por cabo Ethernet.

Tabela 7.3: Tabela de Requisitos Funcionais

Tempo médio (ms)	**Menor tempo (ms)**	**Maior tempo (ms)**
810	582	1026

Comparando com a hipótese formulada, o tempo médio deve ser menor que 1 segundos, o tempo médio de execução é de 810 ms. Isto demonstra que o sistema cumpre com a hipótese definida ao respeitar o tempo limite de 1 segundos.

Capítulo 8

Conclusão

O objetivo do projeto aqui documentado era desenvolver uma plataforma de suporte ao processo de *release planning*. Neste capitulo, é realizada uma análise a todo o trabalho desenvolvido, mediante a apresentação dos objetivos concluídos e do trabalho futuro a realizar.

8.1 Objetivos concluídos

A solução implementada ao longo do projeto, deu resposta ao principal objetivo de desenvolver uma plataforma capaz de sugerir uma data para o final do desenvolvimento de uma *release*.

Porém, não foi possível concluir a implementação de alguns requisitos, nomeadamente, os requisitos relativamente à gestão de históricos e relatórios.

De qualquer modo, considera-se que a arquitetura adotada segue os princípios de reutilização e flexibilidade, que permitem o suporte de novas funcionalidades com o mínimo impacto às já implementadas, pelo que, a implementação destas funcionalidades e outras deverá ser relativamente fácil.

Resumindo, a solução final apresenta-se como uma solução capaz de apoiar o processo de *release planning* da ALERT. Isto é, uma solução que integra o dados do Jira com os dados de férias/ausências de recursos humanos e produz uma sugestão para o final do desenvolvimento de uma *release*. Para além disto, a plataforma também apresenta indicadores estatísticos relativos ao progresso de desenvolvimento das *releases*.

8.2 Trabalho futuro

Relativamente ao trabalho futuro, um dos objetivos futuros previstos para esta plataforma é a adição de um módulo para a gestão das férias/folgas dos Recursos Humanos envolvidos no desenvolvimento das *releases* em substituição do atual modelo de gestão que ocorre em ficheiros Excel.

Para além disto, é necessário implementar as funcionalidades de Relatórios e Histórico que não foram implementadas até à data de produção deste documento e expandir a plataforma para lídar com os dados de outras fases do ciclo de vida de uma *release*, por exemplo fase de Quality Control.

Bibliografia

[1] C. Larman e V. R. Basili. «Iterative and incremental developments. a brief history». Em: *Computer* 36.6 (2003), pp. 47–56. doi: 10.1109/MC.2003.1204375.

[2] G. Ruhe e M. O. Saliu. «The art and science of software release planning». Em: *IEEE Software* 22.6 (2005), pp. 47–53. doi: 10.1109/MS.2005.164.

[3] Christof Ebert. *Software product management*. Vol. 22. 1. 2009, pp. 15–19. isbn: 9783642551406. doi: 10.1007/978-3-642-55140-6.

[4] *Software clínico | ALERT-ONLINE.COM - Português*. url: https://www.alert-online.com/pt/presentation (acedido em 12/01/2021).

[5] *Diretório de Associados do Health Cluster Portugal (HCP)*. url: http://www.healthportugal-directory.com/pt/the-portuguese-health-sector (acedido em 12/01/2021).

[6] admin. *System Development Life Cycle – SDLC*. Innorobix Automation Inc. url: https://www.innorobix.com/uncategorized/sdlc/ (acedido em 19/01/2021).

[7] PMI, ed. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. 5ª ed. Newtown Square, PA: Project Management Institute, 2013. isbn: 978-1-935589-67-9.

[8] Atlassian. *What is version control | Atlassian Git Tutorial*. Atlassian. url: https://www.atlassian.com/git/tutorials/what-is-version-control (acedido em 12/01/2021).

[9] *Apache Subversion*. url: https://subversion.apache.org/ (acedido em 12/01/2021).

[10] *Git*. url: https://git-scm.com/ (acedido em 12/01/2021).

[11] Aline Dresch, Daniel Lacerda e Junico Antunes. *Design Science Research: Método de Pesquisa para Avanço da Ciência e Tecnologia*. Jan. de 2015. isbn: 978-85-8260-298-0. doi: 10.13140/2.1.2264.2885.

[12] Alan Hevner et al. «Design Science in Information Systems Research». Em: *Management Information Systems Quarterly* 28 (mar. de 2004), pp. 75–.

[13] Asmaa H. Elsaid, Rashed K. Salem e Hatem M. Abdelkader. «Proposed framework for planning software releases using fuzzy rule-based system». Em: *IET Software* 13.6 (2019), pp. 543–554. issn: 17518806. doi: 10.1049/iet-sen.2018.5047.

[14] Sami Jantunen et al. «The challenge of release planning: Visible but not seen?» Em: *2011 5th International Workshop on Software Product Management, IWSPM 2011 - Part of the 19th IEEE International Requirements Engineering Conference* (2011), pp. 36–45. doi: 10.1109/IWSPM.2011.6046202.

[15] Amir Seyed Danesh. «Software release planning challenges in software development: An empirical study». Em: *African Journal of Business Management* 6.3 (2012), pp. 956–970. issn: 1993-8233. doi: 10.5897/ajbm11.1519.

[16] zed. *APCER - ISO 9001*. url: https://apcergroup.com/pt/certificacao/pesquisa-de-normas/81/iso-9001 (acedido em 04/02/2021).

[17] Günther Ruhe, Mark Stanford et al. «Intelligent support for software release planning». Em: (2004), pp. 248–262.

[18] G. Ruhe e J. Momoh. «Strategic Release Planning and Evaluation of Operational Feasibility». Em: (2005), 313b–313b. doi: 10.1109/HICSS.2005.561.

[19] A. Al-Emran, D. Pfahl e G. Ruhe. «Decision Support for Product Release Planning Based on Robustness Analysis». Em: (2010), pp. 157–166. doi: 10.1109/RE.2010.28.

[20] A. Ngo-The e G. Ruhe. «Optimized Resource Allocation for Software Release Planning». Em: *IEEE Transactions on Software Engineering* 35.1 (2009), pp. 109–123. doi: 10.1109/TSE.2008.80.

[21] C. Li et al. «An integrated approach for requirement selection and scheduling in software release planning». Em: *Requirements Engineering* 15 (2010), pp. 375–396.

[22] Ph.D. E. William East P.E., F.ASCE. *Critical Path Method (CPM) Tutor for Construction Planning and Scheduling*. en. New York: McGraw-Hill Education, 2015. isbn: 9780071849234. url: https://www.accessengineeringlibrary.com/content/book/9780071849234.

[23] Sofjan H. Nasution. «Fuzzy Critical Path Method». Em: *IEEE Transactions on Systems, Man and Cybernetics* 24.1 (1994), pp. 48–57. issn: 21682909. doi: 10.1109/21.259685.

[24] Wayne D Cottrell. «Simplified program evaluation and review technique (PERT)». Em: *Journal of construction Engineering and Management* 125.1 (1999), pp. 16–22.

[25] Pablo Ballesteros-Pérez. «M-PERT: Manual project-duration estimation technique for teaching scheduling basics». Em: *Journal of construction engineering and management* 143.9 (2017), p. 04017063.

[26] Fernando Nogueira. *Pesquisa Operacional - PERT/CPM*.

[27] Alexandra B. Tenera. «Critical chain buffer sizing: a comparative study». Em: *PMI® Research Conference: Defining the Future of Project Management*. Warsaw, Poland: Project Management Institute, 2008.

[28] Tzvi Raz, Robert Barnes e Dov Dvir. «A Critical Look at Critical Chain Project Management». Em: *Project Management Journal* 34.4 (2003), pp. 24–32. doi: 10.1177/875697280303400404. eprint: https://doi.org/10.1177/875697280303400404. url: https://doi.org/10.1177/875697280303400404.

[29] *Agile Scrum Software - Careers | Axosoft*. url: https://www.axosoft.com/.

[30] *Online Gantt Chart Software and Project Planning Tool | TeamGantt*. url: https://www.teamgantt.com/h.

[31] *Teamwork: Work and Project Management Software*. url: https://www.teamwork.com/.

[32] *VersionOne | CollabNet VersionOne*. url: https://www.collab.net/products/versionone.

[33] *Versatile and Robust Project Management Software | Wrike*. url: https://www.wrike.com/.

[34] G.David Hughes e Don C. Chafin. «Turning new product development into a continuous learning process». Em: *Journal of Product Innovation Management* 13.2 (1996), pp. 89–104. issn: 0737-6782. doi: https://doi.org/10.1016/0737-6782(95)00112-3. url: https://www.sciencedirect.com/science/article/pii/0737678295001123.

[35] Lawrence D Miles. *Techniques of value analysis and engineering*. Miles Value Foundation, 2015.

[36] P. Koen et al. «Fuzzy Front End : Effective Methods , Tools , and Techniques». Em: 2002.

[37] Kuniyoshi Urabe, John Child e Tadao Kagono. *Innovation and management: international comparisons*. W. de Gruyter, 1988.

[38] P. Koen, Heidi M. J. Bertels e E. Kleinschmidt. «Managing the Front End of Innovation—Part I: Results From a Three-Year Study». Em: *Research-Technology Management* 57 (2014), pp. 34–43.

[39] Peter Koen et al. «Providing Clarity and A Common Language to the "Fuzzy Front End"». Em: *Research-Technology Management* 44.2 (2001), pp. 46–55. doi: 10.1080/08956308.2001.11671418. eprint: https://doi.org/10.1080/08956308.2001.11671418. url: https://doi.org/10.1080/08956308.2001.11671418.

[40] Tony Woodall. «Conceptualising 'Value for the Customer': An Attributional, Structural and Dispositional Analysis». Em: *Academy of Marketing Science Review* 12 (jan. de 2003).

[41] Raquel Fernández e M. Bonillo. «The concept of perceived value: A systematic review of the research». Em: *Marketing Theory - MARK THEORY* 7 (dez. de 2007), pp. 427–451. doi: 10.1177/1470593107083165.

[42] Ajit Kambil et al. «Re-Inventing Value Propositions». Em: (1996).

[43] Martin Fowler. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch*. Addison-Wesley, 2012.

[44] Jim Arlow e Ila Neustadt. *UML 2 and the unified process: practical object-oriented analysis and design*. Pearson Education, 2005.

[45] C.E. Vazquez e G.S. Simões. *Engenharia de Requisitos: software orientado ao negócio*. BRASPORT, 2016. isbn: 9788574527901.

[46] K. Wiegers e J. Beatty. *Software Requirements*. Developer Best Practices. Pearson Education, 2013. isbn: 9780735679627. url: https://books.google.pt/books?id=nbpCAwAAQBAJ.

[47] Omg. *An OMG ® Unified Modeling Language ® Publication OMG ® Unified Modeling Language ® (OMG UML ®) OMG Document Number: Date*. 2009. url: https://www.omg.org/spec/UML/20161101/PrimitiveTypes.xmi.

[48] *Think you've got your requirements defined? Think FURPS! - Professional Services Plus*. url: https://www.psplus.ca/articles/think-youve-got-your-requirements-defined-think-furps/.

[49] *Jira | Issue and Project Tracking Software | Atlassian*. url: https://www.atlassian.com/software/jira.

[50] *Integrating with Jira Software Server*. url: https://developer.atlassian.com/server/jira/platform/integrating-with-jira-software-server/.

[51] *Webhooks*. url: https://developer.atlassian.com/server/jira/platform/webhooks/.

[52] Len Bass, Paul Clements e Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[53] Philippe Kruchten. *Architectural Blueprints-The "4+1"View Model of Software Architecture*. 1995, pp. 42–50.

[54] *Autenticação vs. autorização - Microsoft identity platform | Microsoft Docs*. url: https://docs.microsoft.com/pt-pt/azure/active-directory/develop/authentication-vs-authorization.

[55] Julian Reschke. *The 'Basic' HTTP Authentication Scheme*. RFC 7617. Set. de 2015. doi: 10.17487/RFC7617. url: https://rfc-editor.org/rfc/rfc7617.txt.

[56] *Oracle Database Express Edition | Oracle Portugal*. url: https://www.oracle.com/pt/database/technologies/appdev/xe.html.

[57] Dorota Huizinga e Adam Kolawa. *Automated defect prevention: best practices in software management*. John Wiley & Sons, 2007.

[58] *JUnit 5*. url: https://junit.org/junit5/.

[59] Martyn A Ould e Charles Unwin. *Testing in software development*. Cambridge University Press, 1986.

[60] *Postman | The Collaboration Platform for API Development*. url: `https://www.postman.com/`.

[61] T.L. Saaty. *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. Advanced book program. McGraw-Hill International Book Company, 1980. isbn: 9780070543713. url: `https://books.google.pt/books?id=Xxi7AAAAIAAJ`.

Apêndice A

Analytic Hierarchy Process

O método AHP foi introduzido por Thomas Saaty como uma ferramenta eficiente para lidar com a decisões complexas. É um método multi-critério e contempla sete fases: divisão hierarquia, comparação entre critérios, definição de prioridades relativas, avaliação da consistência das prioridades relativas, construção da matriz de comparação paritária para cada critério, definição de prioridades relativas de cada ideia e no final, escolha da ideia [61].

A.1 Fase 1 - Divisão Hierarquia

A divisão hierarquia consiste na construção de uma árvore hierárquica de decisão onde se irão encontrar os objetivos, os critérios e as alternativas viáveis. Para que, através de uma árvore hierárquica, seja possível escolhe a melhor ideia de projeto, é necessário que os critérios estejam bem estruturados.

A Figura A.1 ilustra a árvore estruturada para selecionar a melhor ideia para o projeto.

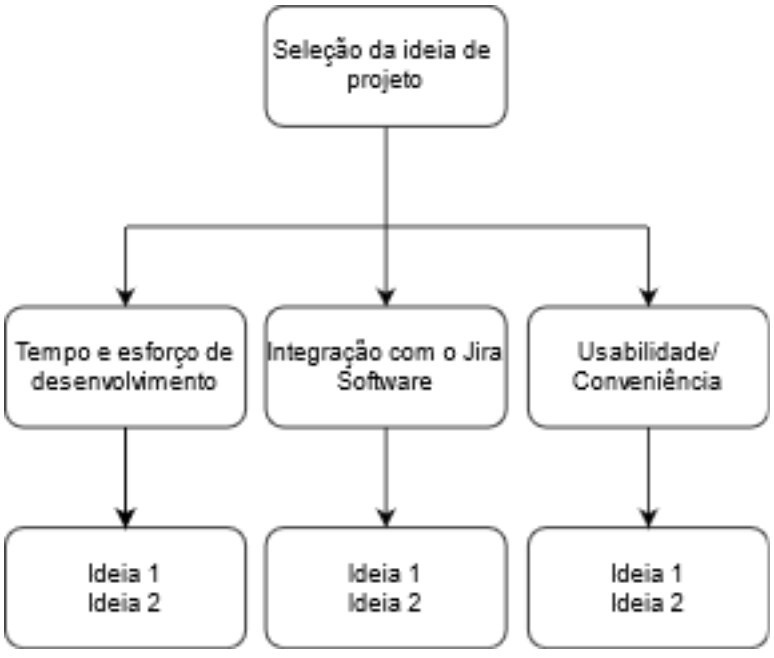

Figura A.1: Árvore hierárquica de decisão dividida pelos critérios

A aplicação do método teve por base três fatores/critérios: Tempo e esforço de desenvolvimento, Integração com o Jira *Software* e Usabilidade/Conveniência.

Os critérios utilizados (Tempo e esforço de desenvolvimento, Integração com o Jira *Software*, Bom desempenho e Usabilidade/Conveniência) foram escolhidos com o intuito de apoiar na decisão de qual das ideias geradas inicialmente devia ser a escolhida.

- **Tempo e esforço de desenvolvimento** - Em qualquer projeto o tempo e o esforço de desenvolvimento é sempre um critério importante. Um elevado tempo de desenvolvimento é sempre indesejado. Quando menor for o tempo de desenvolvimento e esforço necessário mais tempo sobra para outras tarefas cruciais como testes, desenvolvimento de funcionalidades não prioritárias, maior dedicação para com o resquerimentos não funcionais, etc...
- **Integração com o Jira Software** - É um dos principais objetivos da plataforma. Sem esta integração a plataforma não tem grande utilidade, pois é através desta integração que se irão obter as tarefas a escalonar pelos recursos humanos, para que, assim, se possa fazer as devidas estimativas para a *release*. É também, através da integração, que se irão obter os dados estatísticos fundamentais para o acompanhamento do desenvolvimento da *release* por parte equipa de *Release Planning*.
- **Usabilidade/Conveniência** - Mais dois aspetos importantes deste projeto. A usabilidade deve ser boa e a conveniência também deve tida em conta, caso contrário a aplicação não irá cumprir o propósito de auxiliar a equipa de *Release Planning*.

A.2 Fase 2 - Comparação entre critérios

Nesta etapa, as prioridades entre os elementos de cada nível da hierarquia são estabelecidas com recurso a uma matriz de comparação.

Um dos aspetos chave a ser considerados para a criação dessa matriz é a determinação de escala de valores que não deve ultrapassar um total de nova fatores, com o intuito de manter a consistência da matriz. A escala, definida por Saaty, é chamada de Escala Fundamental e pode ser consultada abaixo em A.2.

Nivel de importância	Definição	Explicação
1	Igual importânica	As duas atividades contribuem igualmente para o objetivo
3	Fraca importância	A experiência e o julgamento favorecem levemente uma atividade em relação à outra
5	Forte importância	A experiência e o julgamento favorecem fortemente uma atividade em relação à outra
7	Muito forte importância	Uma atividade é muito fortemente favorecida em relação a outra
9	Importância absoluta	A evidência favorece uma atividade em relação a outra com o mais alto grau de certeza
2,4,6,8	Valores intermediários	Quando se procura uma condição de compromisso entre duas definições

Figura A.2: Escala Fundamental

Com vista a obter uma das duas ideias, através dos critérios definidos previamente na divisão hierárquica, a seguinte matriz (A.1) de comparação foi desenvolvida.

Tabela A.1: Matriz de comparação de critérios

	A	**B**	**C**
A	1	1/5	4
B	5	1	9
C	1/4	1/9	1

Sendo:

- **A**: Tempo e esforço de desenvolvimento
- **B**: Integração com o Jira *Software*
- **C**: Usabilidade/Conveniência

A.3 Fase 3 - Definição de Prioridades Relativas

Estabelecidas as prioridades entre os critérios é necessário obter as suas prioridades relativas. As prioridades relativas têm como objetivo identificar a ordem de importância de cada critério. Para obter as prioridades relativas é necessário, em primeiro lugar, normalizar a matriz da comparação de critérios, e, em seguida, efectuar a média aritmética dos valores dessa matriz.

As prioridades relativas para os critérios previamente definidos estão apresentados na Tabela A.2.

Tabela A.2: Prioridades Relativas dos Critérios

Critério	Prioridade Relativa
A	0.1994
B	0.7352
C	0.0654

Assim, o critério **Integração com o Jira Software** é o mais importante, seguindo-se o critério **Tempo e esforço de desenvolvimento** e, por fim, o critério **Usabilidade/Conveniência**.

A.4 Fase 4 - Avaliação da Consistência das Prioridades Relativas

Com vista a medir a consistência dos julgamentos foram em relação a grandes amostras de juízos completamente aleatórios, efectuou-se o calculo da Razão de Consistência (RC).

$$RC = \frac{IC}{IR} \tag{A.1}$$

A RC (A.1) obtém-se através da divisão do valor de Índice de Consistência (IC) com o valor de Índice Randómico (IR) em que o IC é obtido pela seguinte equação:

$$IC = \frac{\lambda_{max} - n}{n - 1} \tag{A.2}$$

E IR é obtido através da consulta de uma tabela proposta por Satty que reúne os IRs de matrizes de ordem 1 a 15 (A.3).

Tabela A.3: IR Médio do AHP [Fonte: [61]]

1	2	3	4	5	6	7	8
0.00	0.00	0.58	0.90	1.12	1.24	1.32	1.41

9	10	11	12	13	14	15
1.45	1.49	1.51	1.48	1.56	1.57	1.59

A obtenção do valor da RC começa, então, pelo calculo do valor próprio da matriz de comparação de critérios (A.1), λ_{max}, através da seguinte equação:

$$Ax = \lambda_{max} x, \tag{A.3}$$

onde A é a matriz e x as prioridades relativas.

Assim sendo,

$$\begin{bmatrix} 1 & 1/5 & 4 \\ 5 & 1 & 9 \\ 1/4 & 1/9 & 1 \end{bmatrix} \begin{bmatrix} 0.1994 \\ 0.7352 \\ 0.0654 \end{bmatrix} \cong \lambda_{max} \begin{bmatrix} 0.1994 \\ 0.7352 \\ 0.0654 \end{bmatrix} \Leftrightarrow \begin{bmatrix} 0.6080 \\ 2.3208 \\ 0.1969 \end{bmatrix} \cong \lambda_{max} \begin{bmatrix} 0.1994 \\ 0.7352 \\ 0.0654 \end{bmatrix} \Leftrightarrow$$
$$\Leftrightarrow \lambda_{max} = average\{\frac{0.6080}{0.1994}, \frac{2.3208}{0.7352}, \frac{0.0654}{0.1969}\} \cong 3.07244462 \quad \text{(A.4)}$$

Tendo sido calculado o valor próprio da matriz de comparação de critérios, procede-se ao calculo do IC. Assim, sendo n o número de critérios, tem-se:

$$IC = \frac{\lambda_{max} - n}{n-1} = \frac{3.69463251 - 3}{3-1} \cong 0.34731625 \quad \text{(A.5)}$$

Obtido o valor do IC resta apenas consultar qual o IR definido para 3 critérios na Tabela A.3 e calcular o RC.

$$RC = \frac{IC}{IR} = \frac{0.34731625}{0.58} \cong 0.59882113 \quad \text{(A.6)}$$

Dado que o $RC \cong 0.6$ e inferior a 0.1, é possível afirmar que os valores atribuídos para as prioridades relativas são consistentes e confiáveis. .

A.5 Fase 5 - Construção da Matriz de Comparação Paritária

Confirmada a consistência das prioridades relativas, segue-se a priorização das ideias geradas, que foram apresentadas em 3.1.2, relativamente a cada critério.

Assumindo que:

- **X** - Implementação de uma plataforma que receba os dados das férias e ausências dos recursos humanos de um ficheiro Excel;
- **Y** - Implementação de um plataforma que fornece-se um módulo de gestão de férias e ausências (marcação de férias, aprovação de férias, etc...).

Foram criadas três matrizes de comparação, uma para o critério **Tempo e esforço de desenvolvimento** outra para o critério **Integração com o Jira Software** e outra para o critério **Usabilidade/Conveniência**.

Em seguida serão apresentadas três tabelas (Tabela A.4; Tabela A.5; Tabela A.6) onde são apresentadas as matrizes de comparação para cada critério e as prioridades relativas.

Tabela A.4: Matriz de comparação paritária de Tempo e esforço de desenvolvimento

	X	Y	Prioridade Relativa
X	1	5	0.83
Y	0.2	1	0.17

Tabela A.5: Matriz de comparação paritária de Integração com o Jira *Software*

	X	Y	Prioridade Relativa
X	1	1	0.5
Y	1	1	0.5

Tabela A.6: Matriz de comparação paritária de Usabilidade/Conveniência

	X	Z	Prioridade Relativa
X	1	1/5	0.17
Y	5	1	0.83

No final, a árvore hierárquica foi atualizada. Assim, é possível obter num só diagrama (Figura A.3) toda a informação recolhida e permite uma melhor preparação para a próxima etapa do AHP

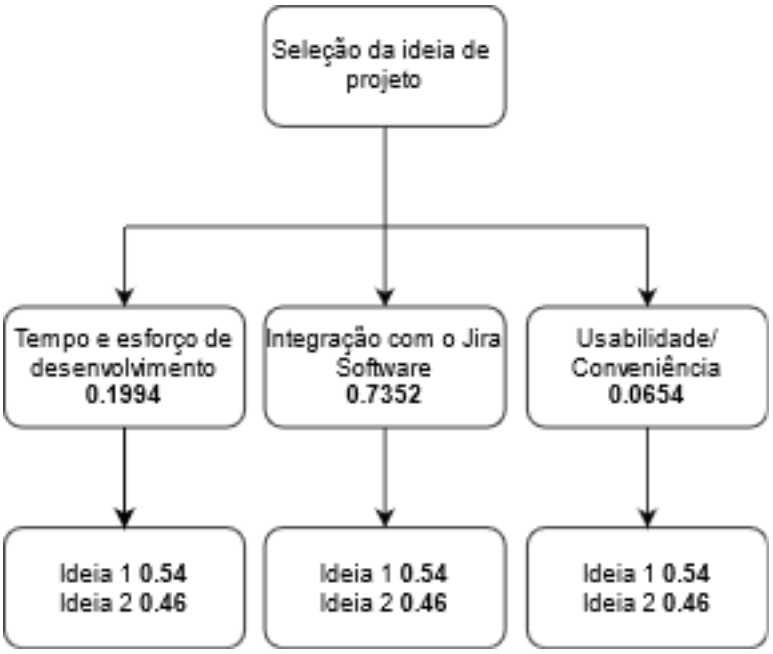

Figura A.3: Árvore hierárquica de decisão atualizada

A.6 Fase 6 - Definição de Prioridades Relativas de cada Ideia

Este passo, antecipa o último passo que é a escolha da ideia, e pretende obter uma matriz de prioridade composta das ideias alternativas previamente apresentadas.

A matriz de prioridade composta é obtida através da multiplicação dos valores de prioridade relativa calculados na fase 5 com o vetor de prioridades dos critérios.

$$\begin{bmatrix} 0.83 & 0.50 & 0.17 \\ 0.17 & 0.50 & 0.83 \end{bmatrix} x \begin{bmatrix} 0.1994 \\ 0.7352 \\ 0.0654 \end{bmatrix} = \begin{bmatrix} 0.54 \\ 0.46 \end{bmatrix} \quad \text{(A.7)}$$

A.7 Fase 7 - Escolha da Ideia

A ideia recomenda pela aplicação do AHP é a **Implementação de uma plataforma que receba os dados das férias e ausências dos recursos humanos de um ficheiro Excel**. Esta conclusão tem por base a análise dos valores da matriz composta, onde se verifica que esta opção tem um peso de 0.54.

Apêndice B

Requisitos Funcionais

B.1 FR-01 - Carregar dados dos Recursos Humanos de um ficheiro Excel

Tabela B.1: Detalhes do requisito FR-01

Descrição	Um membro da equipa de *Release Planning* deseja carregar no sistema um ficheiro Excel com as férias e ausências dos Recursos Humanos.
Prioridade	Alta
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para efectuar o carregamento do ficheiro. **Fluxo de Eventos:** 1. O membro da equipa inicia a sessão do sistema. 2. O sistema disponibiliza as ações possíveis. 3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos. 4. O sistema apresenta a *dashboard*. 5. O membro navega na *dashboard* até encontrar a opção de carregar dados de férias/ausências. 6. O sistema apresenta uma área que permite ao utilizador seleccionar o ficheiro que deseja carregar. 7. O membro da equipa seleciona o ficheiro que deseja carregar e submete o ficheiro. 8. O sistema apresenta uma mensagem de sucesso. **Pós-condições:** • Os dados do ficheiro são validados pelo sistema. • Os dados do ficheiro são persistidos numa base de dados. • O sistema verifica se houve alteração aos dados previamente existentes no sistema, caso isso se verifique, o sistema atualiza os dados e persiste um histórico dos dados antigos.

Validações:

- As datas de férias/ausências não podem incluir fins-de-semana nem feriados;
- A data de inicio de um período de férias/ausências não pode ser após a data de fim.
- Os dias de férias de um determinado Recurso Humano não pode exceder o limite de férias que esse Recurso Humano tem para um determinado ano civil.

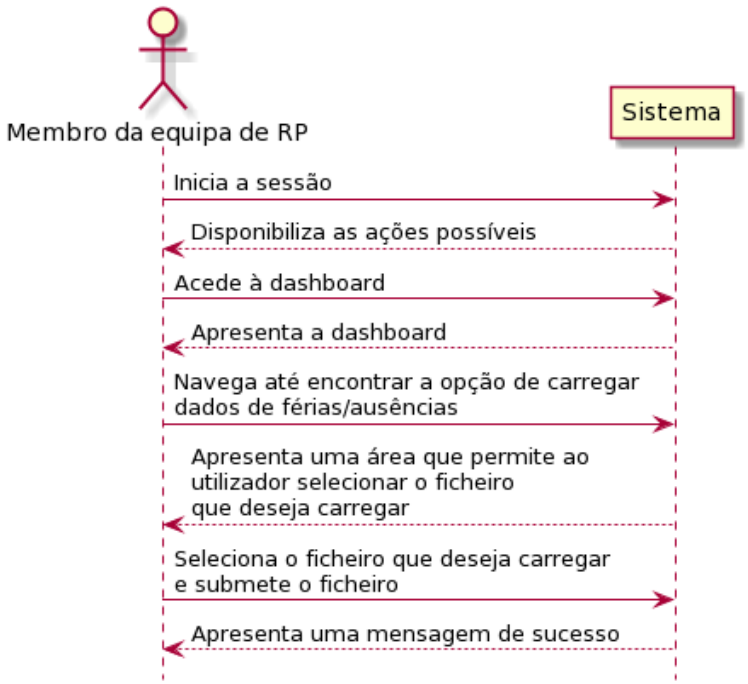

Figura B.1: SSD FR-01

B.2 FR-02 - Listar os Recursos Humanos

Tabela B.2: Detalhes do requisito FR-02

Descrição	Um membro da equipa de *Release Planning* listar os Recursos Humanos que estão guardados no sistema.
Prioridade	Alta
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para efectuar a ação de listar os recursos humanos. • O sistema deve ter Recursos Humanos previamente inseridos.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* até encontrar a opção de listar os Recursos Humanos guardados no sistema.
6. O sistema apresenta a lista dos Recursos Humanos com as suas informações mais relevantes.

Pós-condições:

- n/a

Validações:

- n/a

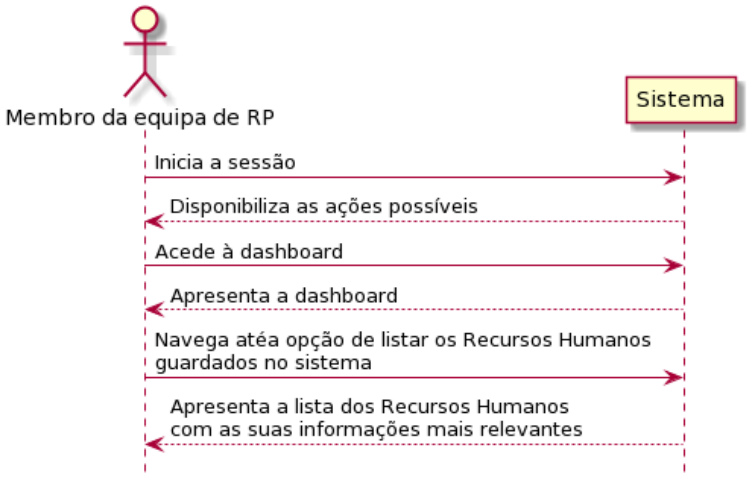

Figura B.2: SSD FR-02

B.3 FR-03 - Alterar o nível de experiência de um Recurso Humano

Tabela B.3: Detalhes do requisito FR-03

Descrição	Um membro da equipa de *Release Planning* altera o nível de experiência de um Recurso Humano guardado no sistema.
Prioridade	Alta

Simulação/Sequência de Respostas

Pré-condições:

- O membro da equipa deve ter permissões para efectuar a ação de alterar o nível de experiência de um Recurso Humano.
- O sistema deve ter Recursos Humanos previamente inseridos.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* até encontrar a opção de listar os Recursos Humanos guardados no sistema.
6. O sistema apresenta a lista dos Recursos Humanos com as suas informações mais relevantes.
7. O membro da equipa seleciona o Recurso Humano que deseja alterar o nível de experiência.
8. O sistema apresenta as ações que pode executar naquele Recurso Huamno.
9. O membro da equipa navega até à opção que lhe permite alterar o nível de expeiência.
10. O sistema apresenta os níveis de experiência possíveis.
11. O membro da equipa escolhe um nível e pressiona guardar.
12. O sistema questiona se o membro da equipa deseja confirmar a ação.
13. O membro da equipa confirma a ação.
14. O sistema atualiza o nível de experiência do Recurso Humano e informa sucesso.

Pós-condições:

- O Recurso Humano fica com um novo nível de experiência a si associado.

Validações:

- n/a

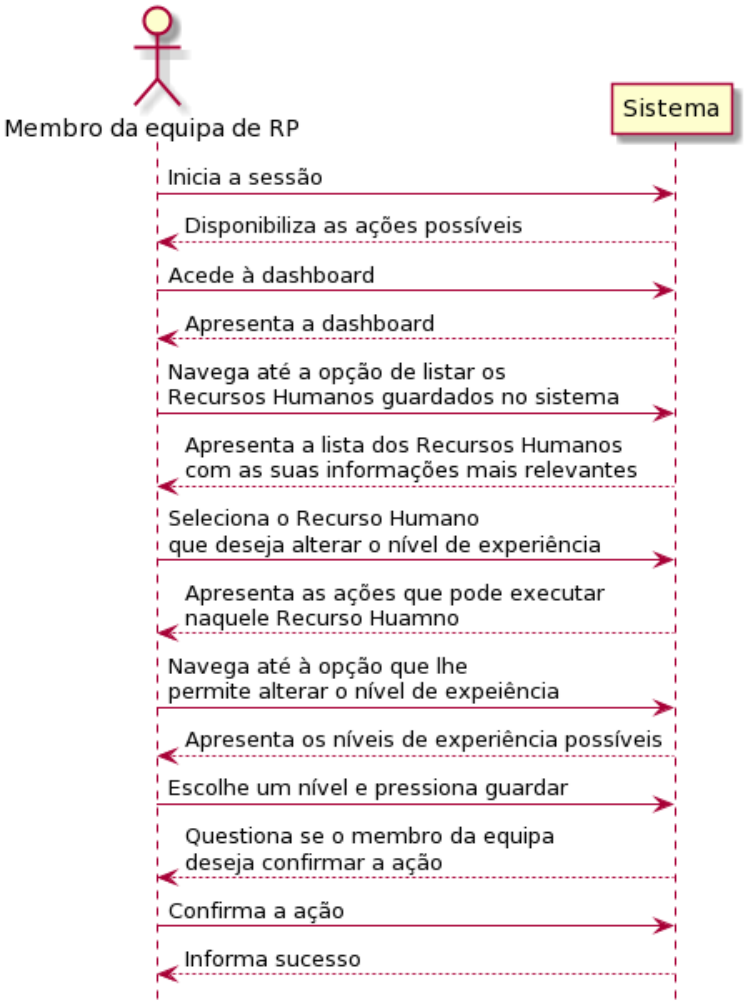

Figura B.3: SSD FR-03

B.4 FR-04 - Definir a percentagem de alocação de um Recurso Humano a uma versão

Tabela B.4: Detalhes do requisito FR-04

Descrição	Um membro da equipa de *Release Planning* deseja definir a percentagem de alocação de um Recurso Humano a uma determinada versão.
Prioridade	Alta

Simulação/Sequência de Respostas

Pré-condições:

- O membro da equipa deve ter permissões para efectuar a ação de definir a percentagem de alocação de um Recurso Humano.
- O sistema deve ter Recursos Humanos previamente inseridos.
- O sistema deve ter Versões previamente inseridas.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* até encontrar a opção de listar os Recursos Humanos guardados no sistema.
6. O sistema apresenta a lista dos Recursos Humanos com as suas informações mais relevantes.
7. O membro da equipa seleciona o Recurso Humano que deseja definir a alocação numa versão.
8. O sistema apresenta uma lista de versões às quais o recurso humano está associado.
9. O membro da equipa escolhe a versão à qual pertende alterar a percentagem de alocação daquele Recurso Humano.
10. O sistema apresenta uma área para introduzir a nova percentagem de alocação para aquele recurso naquela versão.
11. O membro da equipa introduz uma novo valor para a alocação.
12. O sistema questiona se o membro da equipa deseja confirmar a ação.
13. O membro da equipa confirma a ação.
14. O sistema atualiza o valor de alocação do recurso humano aquela versão.

Pós-condições:

- O Recurso Humano fica com uma nova percentagem de alocação a uma determinada versão.
- O sistema recalcula a estimativa para o final do desenvolvimento da versão com base nas alterações efectuadas.

Validações:

- A percentagem de alocação deve estar compreendida no seguinte intervalo de valores 0 e 100, inclusive.

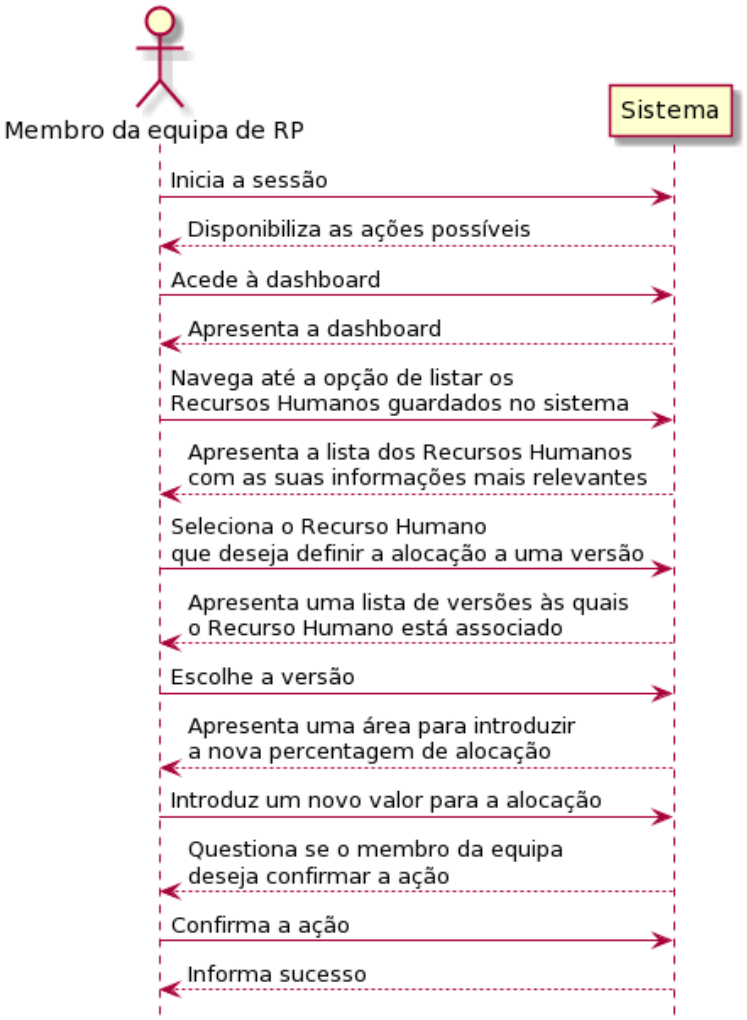

Figura B.4: SSD FR-04

B.5 FR-05 - Consultar as disponibilidades de um Recurso Humano

Tabela B.5: Detalhes do requisito FR-05

Descrição	Um membro da equipa de *Release Planning* deseja consultar as disponibilidades de um Recurso Humano.
Prioridade	Média
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para efectuar a consulta das disponibilidades de um Recurso Humano. • O sistema deve ter Recursos Humanos previamente inseridos.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* até encontrar a opção de listar os Recursos Humanos guardados no sistema.
6. O sistema apresenta a lista dos Recursos Humanos com as suas informações mais relevantes.
7. O membro da equipa seleciona o Recurso Humano que deseja consultar a disponibilidade.
8. O sistema apresenta as informações de disponibilidade para aquele Recurso Humano.

Pós-condições:

- n/a

Validações:

- n/a

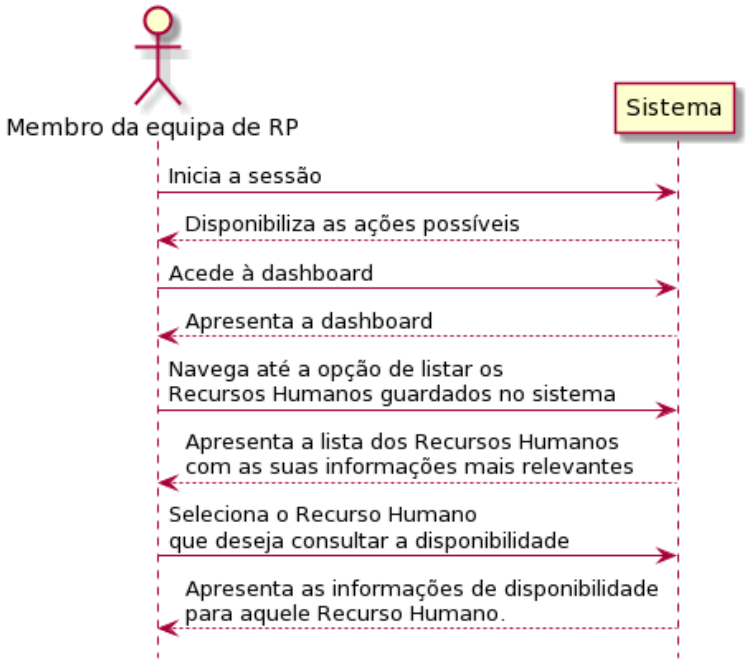

Figura B.5: SSD FR-05

B.6 FR-06 - Criar uma versão

Tabela B.6: Detalhes do requisito FR-06

Descrição	A criação de uma versão na plataforma Jira deve despoletar a criação de uma versão no sistema. Esta versão contém dados (os dados) provenientes do Jira.
Prioridade	Alta
Simulação/Sequência de Respostas	**Pré-condições:** • O sistema deve estar integrado com a plataforma Jira. **Fluxo de Eventos:** 1. O Jira envia uma mensagem em formato JSON para o sistema. 2. O sistema cria uma versão com os dados da mensagem. **Pós-condições:** • Uma versão é criada e guardada no sistema. • A estimativa para o final do desenvolvimento da versão é calculado e guardado no sistema. **Validações:** • n/a

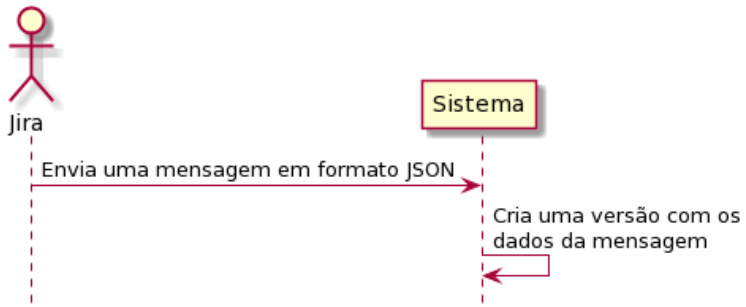

Figura B.6: SSD FR-06

B.7 FR-07 - Alterar uma versão

Tabela B.7: Detalhes do requisito FR-07

Descrição	Alterações às versões do Jira devem despoletar alterações às versões do sistema.
Prioridade	Alta

Simulação/Sequência de Respostas	**Pré-condições:** • O sistema deve estar integrado com a plataforma Jira. • A versão do Jira deve coincidir com uma versão do sistema. **Fluxo de Eventos:** 1. O Jira envia uma mensagem em formato JSON para o sistema. 2. O sistema altera a versão de acordo com os dados da mensagem. **Pós-condições:** • A versão é alterada. • A estimativa para o final do desenvolvimento da versão é calculado e guardado no sistema. • As alterações à versão são registadas no histórico da versão. **Validações:** • A versão proveniente do Jira deve coincidir com uma versão guardada no sistema.

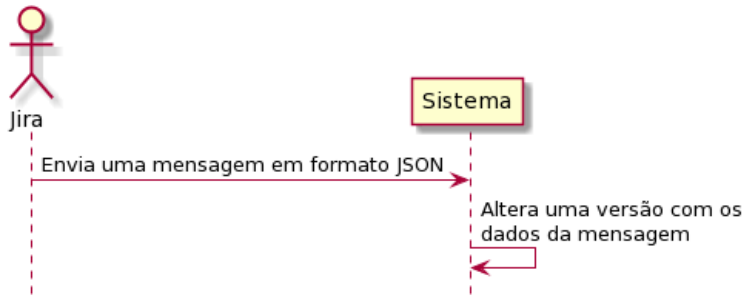

Figura B.7: SSD FR-07

B.8 FR-08 - Adicionar issue de uma versão

Tabela B.8: Detalhes do requisito FR-08

Descrição	Criação de *issues* no Jira despoleta um evento que é enviado para o sistema.
Prioridade	Alta

Simulação/Sequência de Respostas	**Pré-condições:** • O sistema deve estar integrado com a plataforma Jira. • O *issue* criado deve pertencer a uma versão existente no sistema. **Fluxo de Eventos:** 1. O Jira envia uma mensagem em formato JSON para o sistema. 2. O sistema cria um *issue* de acordo com os dados da mensagem. **Pós-condições:** • Um novo *issue* é criado. • A estimativa para o final do desenvolvimento da versão é calculado e guardada no sistema. • Alterações à versão são registadas no histórico da versão. **Validações:** • A versão do *issue* proveniente do Jira deve coincidir com uma versão guardada no sistema.

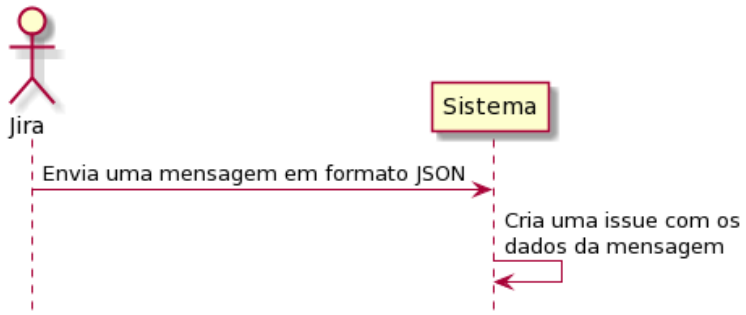

Figura B.8: SSD FR-08

B.9 FR-09 - Atualizar issue de uma versão

Tabela B.9: Detalhes do requisito FR-09

Descrição	A alteração de *issues* no Jira despoleta um evento que é enviado para o sistema.
Prioridade	Alta

Simulação/Sequência de Respostas	**Pré-condições:** • O sistema deve estar integrado com a plataforma Jira. • O *issue* modificado deve pertencer a uma versão existente no sistema. **Fluxo de Eventos:** 1. O Jira envia uma mensagem em formato JSON para o sistema. 2. O sistema atualiza o *issue* de acordo com os dados da mensagem. **Pós-condições:** • Um *issue* é atualizado. • A estimativa para o final do desenvolvimento da versão é calculado e guardada no sistema. • Alterações à versão são registadas no histórico da versão. **Validações:** • A versão do *issue* proveniente do Jira deve coincidir com uma versão guardada no sistema.

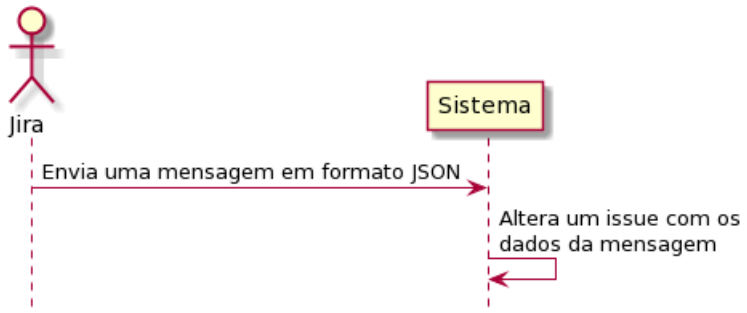

Figura B.9: SSD FR-09

B.10 FR-10 - Apagar issue de uma versão

Tabela B.10: Detalhes do requisito FR-09

Descrição	A eliminação do valor do campo "Planned Version"do *issues* no Jira despoleta um evento que é enviado para o sistema.
Prioridade	Alta

Simulação/Sequência de Respostas	**Pré-condições:** • O sistema deve estar integrado com a plataforma Jira. • O *issue* modificado deve pertencer a uma versão existente no sistema. **Fluxo de Eventos:** 1. O Jira envia uma mensagem em formato JSON para o sistema. 2. O sistema apaga o *issue* da versão onde estava. **Pós-condições:** • Um *issue* é atualizado. • A estimativa para o final do desenvolvimento da versão é calculado e guardada no sistema. • Alterações à versão são registadas no histórico da versão. **Validações:** • O *issue* proveniente do Jira deve coincidir com um *issue* guardado no sistema.

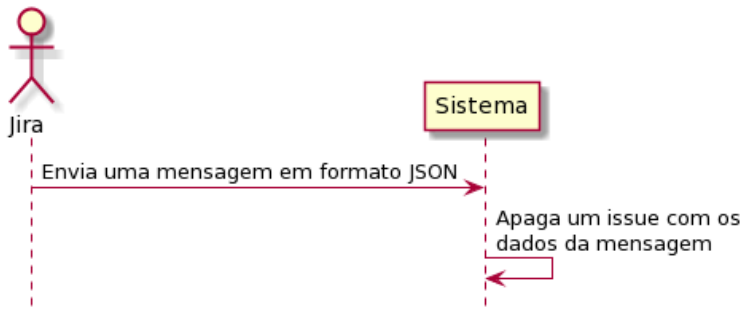

Figura B.10: SSD FR-10

B.11 FR-11 - Listar as versões

Tabela B.11: Detalhes do requisito FR-08

Descrição	Um utilizador (membro da equipa de *Release Planning* ou um membro da equipa de desenvolvimento) deseja listar as versões.
Prioridade	Alta

Simulação/Sequência de Respostas

Pré-condições:

- O utilizador deve ter permissões para listar versões.
- Devem existir versões guardadas no sistema.

Fluxo de Eventos:

1. O utilizador inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de versões.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* selecionando a opção de listar versões.
6. O sistema apresenta uma lista de versões.

Pós-condições:

- n/a

Validações:

- n/a

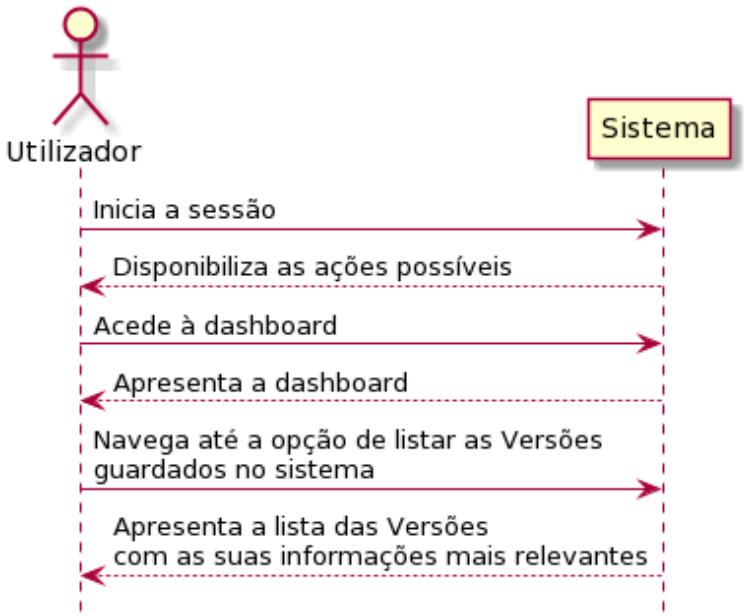

Figura B.11: SSD FR-11

B.12 FR-12 - Consultar os indicadores estatísticos de uma versão

Tabela B.12: Detalhes do requisito FR-09

Descrição	Um utilizador (membro da equipa de *Release Planning* ou um membro da equipa de desenvolvimento) deseja consultar os indicadores estatísticos de uma versão.
Prioridade	Alta
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para efectuar a ação de apagar o histórico de uma versão. • O sistema deve ter realizado o calculo para determinar uma estimativa para o final do desenvolvimento da versão. • O sistema deve ter calculado o progresso do desenvolvimento da versão. • O sistema deve ser capaz de realizar o calculo dos indicadores estatísticos. **Fluxo de Eventos:** 1. O membro da equipa inicia a sessão do sistema. 2. O sistema disponibiliza as ações possíveis. 3. O membro da equipa acede à *dashboard* dedicada para a gestão de versões. 4. O sistema apresenta a *dashboard*. 5. O membro da equipa navega na *dashboard* selecionando a opção de listar versões. 6. O sistema apresenta uma lista de versões. 7. O membro da equipa seleciona uma versão. 8. O sistema apresenta os dados estatísticos da versão. **Pós-condições:** • n/a **Validações:** • n/a

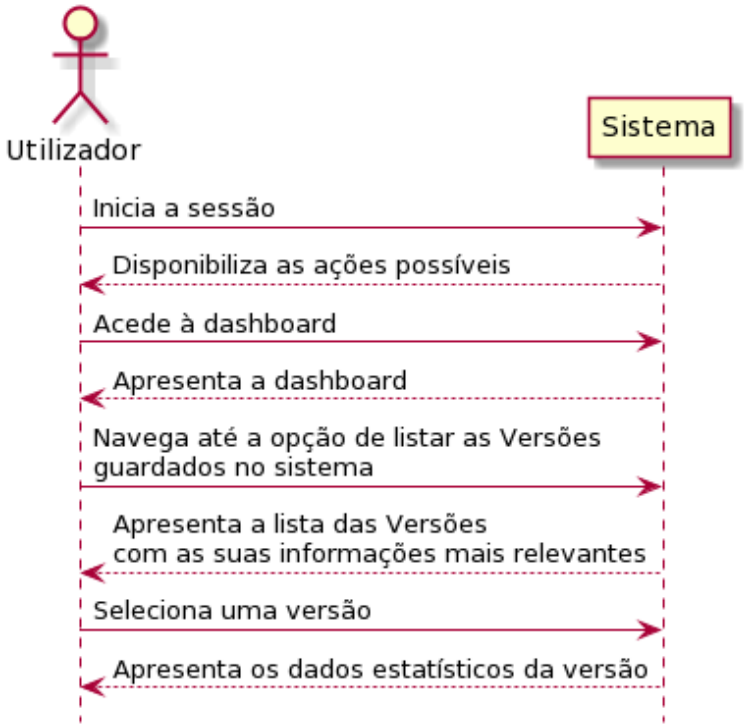

Figura B.12: SSD FR-12

B.13 FR-13 - Fazer o download de relatórios de uma versão

Tabela B.13: Detalhes do requisito FR-10

Descrição	Um membro da equipa de *Release Planning* deseja descarregar relatórios sumários de uma determinada versão.
Prioridade	Média
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para efectuar a ação de descarregar relatórios sumários de uma determinada versão. • A versão da qual deseja obter os relatórios deve estar concluída. • O sistema deve conseguir gerar os relatórios com sucesso.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de relatórios.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* até encontrar a opção relatórios de versões.
6. O sistema apresenta uma lista de relatórios de versões.
7. O membro da equipa seleciona o relatório que deseja descarregar.
8. O sistema descarrega um ficheiro em formato PDF para o computador do membro da equipa e informa sucesso.

Pós-condições:

- n/a

Validações:

- n/a

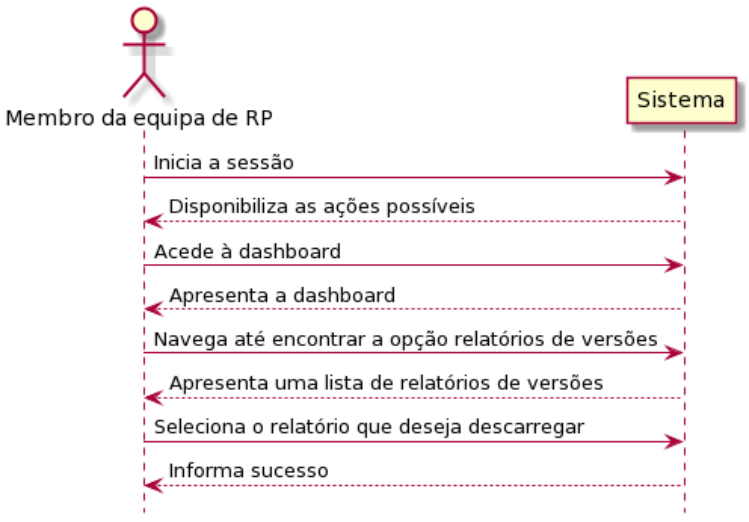

Figura B.13: SSD FR-13

B.14 FR-14 - Consultar o histórico de uma versão

Tabela B.14: Detalhes do requisito FR-11

Descrição	Um membro da equipa de *Release Planning* deseja consultar o histórico de uma versão.

Prioridade	Média
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para consultar o histórico de uma versão. • O histórico da versão deve existir. **Fluxo de Eventos:** 1. O membro da equipa inicia a sessão do sistema. 2. O sistema disponibiliza as ações possíveis. 3. O membro da equipa acede à *dashboard* dedicada para a gestão de versões. 4. O sistema apresenta a *dashboard*. 5. O membro da equipa navega na *dashboard* selecionando a opção de listar versões. 6. O sistema apresenta uma lista de versões. 7. O membro da equipa seleciona uma versão. 8. O sistema apresenta dados da versão e possíveis ações a tomar relativamente aquela versão. 9. O membro da equipa navega até encontrar a opção que lhe permite consultar o histórico daquela versão e seleciona-a. 10. O sistema apresenta o histórico daquela versão. **Pós-condições:** • n/a **Validações:** • n/a

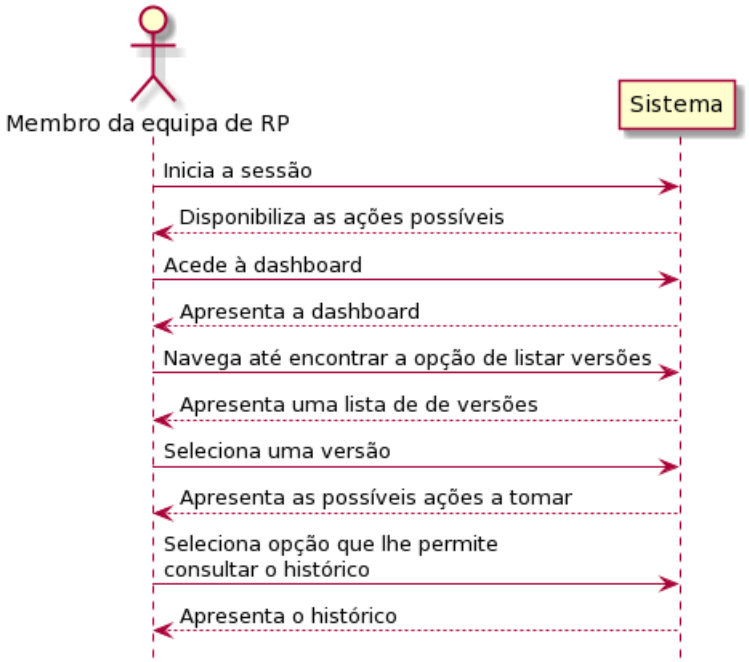

Figura B.14: SSD FR-14

B.15 FR-15 - Apagar o histórico de uma versão

Tabela B.15: Detalhes do requisito FR-12

Descrição	Um membro da equipa de *Release Planning* deseja apagar o histórico de uma versão.
Prioridade	Baixa
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para efectuar a ação de apagar o histórico de uma versão. • O histórico da versão deve existir.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de versões.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* selecionando a opção de listar versões.
6. O sistema apresenta uma lista de versões.
7. O membro da equipa seleciona uma versão.
8. O sistema apresenta dados da versão e possíveis ações a tomar relativamente aquela versão.
9. O membro da equipa navega até encontrar a opção que lhe permite apagar o histórico daquela versão e seleciona-a.
10. O sistema pede confirmação ao membro da equipa.
11. O membro da equipa confirma a ação.
12. O sistema apaga o histórico da versão e mostra sucesso.

Pós-condições:

- O histórico da versão selecionada é apagado do sistema.

Validações:

- n/a

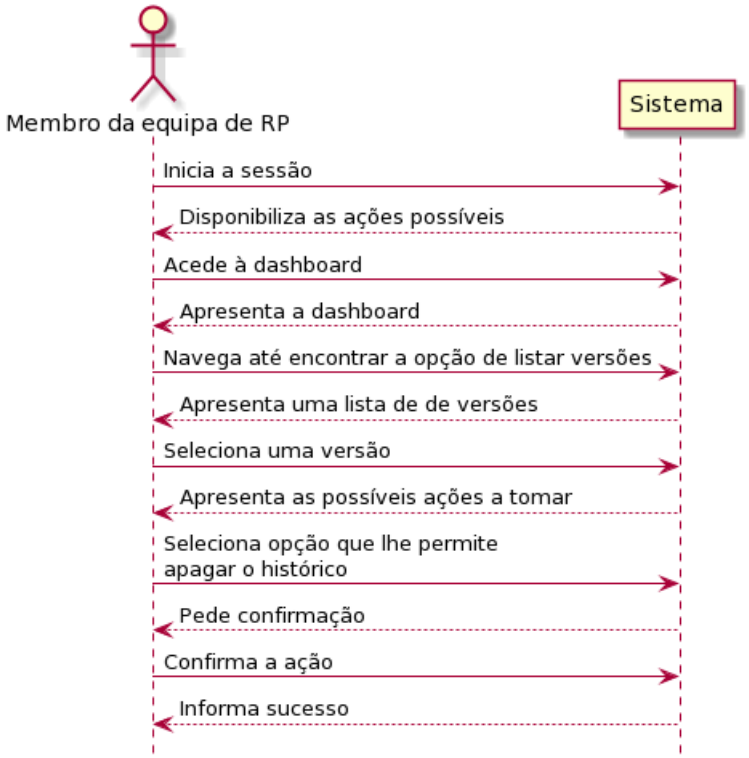

Figura B.15: SSD FR-15

B.16 FR-16 - Consultar o histórico de férias/ausências de um Recurso Humano

Tabela B.16: Detalhes do requisito FR-13

Descrição	Um membro da equipa de *Release Planning* deseja consultar o histórico das alterações que ocorreram relativamente às férias/ausências de um Recurso Humano.
Prioridade	Baixa
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para efectuar a consulta do histórico das férias/ausências de um Recurso Humano. • O histórico das férias/ausências do Recurso Humano deve existir.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* até encontrar a opção de listar os Recursos Humanos guardados no sistema.
6. O sistema apresenta a lista dos Recursos Humanos com as suas informações mais relevantes.
7. O membro da equipa seleciona o Recurso Humano que deseja consulta o histórico de histórico férias/ausências.
8. O sistema apresenta informações e possíveis ações relativas ao Recurso Humano.
9. O membro da equipa navega na página até encontra a opção que lhe permita consulta o histórico de férias/ausências do Recurso Humano pressionando-a opção.
10. O sistema apresenta o histórico de férias/ausências do Recurso Humano.

Pós-condições:

- n/a

Validações:

- n/a

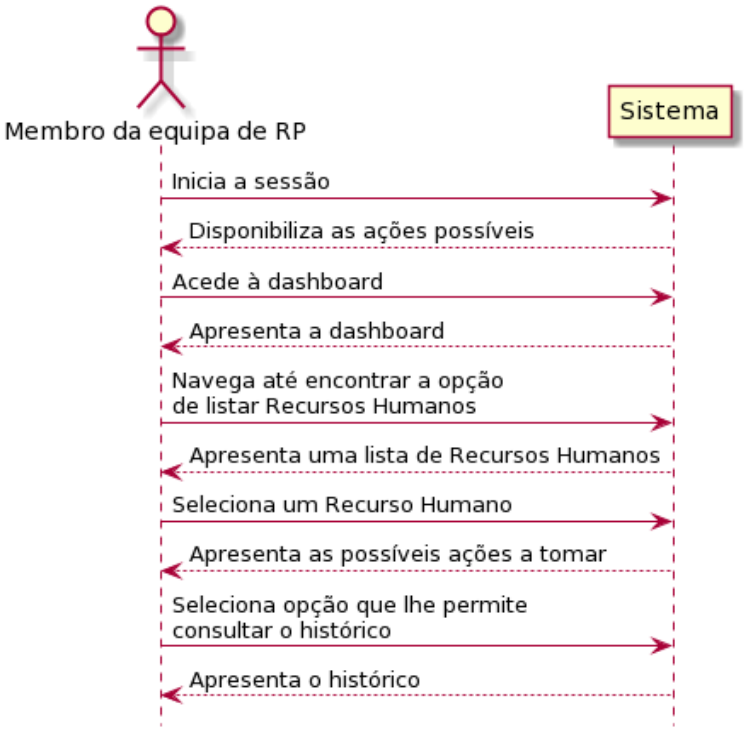

Figura B.16: SSD FR-16

B.17 FR-17 - Apagar o histórico de férias/ausências de um Recurso Humano

Tabela B.17: Detalhes do requisito FR-14

Descrição	Um membro da equipa de *Release Planning* deseja apagar o histórico de uma versão.
Prioridade	Baixa
Simulação/Sequência de Respostas	**Pré-condições:** • O membro da equipa deve ter permissões para apagar o histórico das férias/ausências de um Recurso Humano. • O histórico das férias/ausências do Recurso Humano deve existir.

Fluxo de Eventos:

1. O membro da equipa inicia a sessão do sistema.
2. O sistema disponibiliza as ações possíveis.
3. O membro da equipa acede à *dashboard* dedicada para a gestão de Recursos Humanos.
4. O sistema apresenta a *dashboard*.
5. O membro da equipa navega na *dashboard* até encontrar a opção de listar os Recursos Humanos guardados no sistema.
6. O sistema apresenta a lista dos Recursos Humanos com as suas informações mais relevantes.
7. O membro da equipa seleciona o Recurso Humano que deseja consulta o histórico de histórico férias/ausências.
8. O sistema apresenta informações e possíveis ações relativas ao Recurso Humano.
9. O membro da equipa navega na página até encontra a opção que lhe permita apagar o histórico de férias/ausências do Recurso Humano pressionando-a opção.
10. O sistema pede confirmação ao membro da equipa.
11. O membro da equipa confirma a ação.
12. O sistema apaga o histórico de férias/ausências do Recurso Humano.

Pós-condições:

- O histórico de férias/ausências do Recurso Humano é apagado do sistema.

Validações:

- n/a

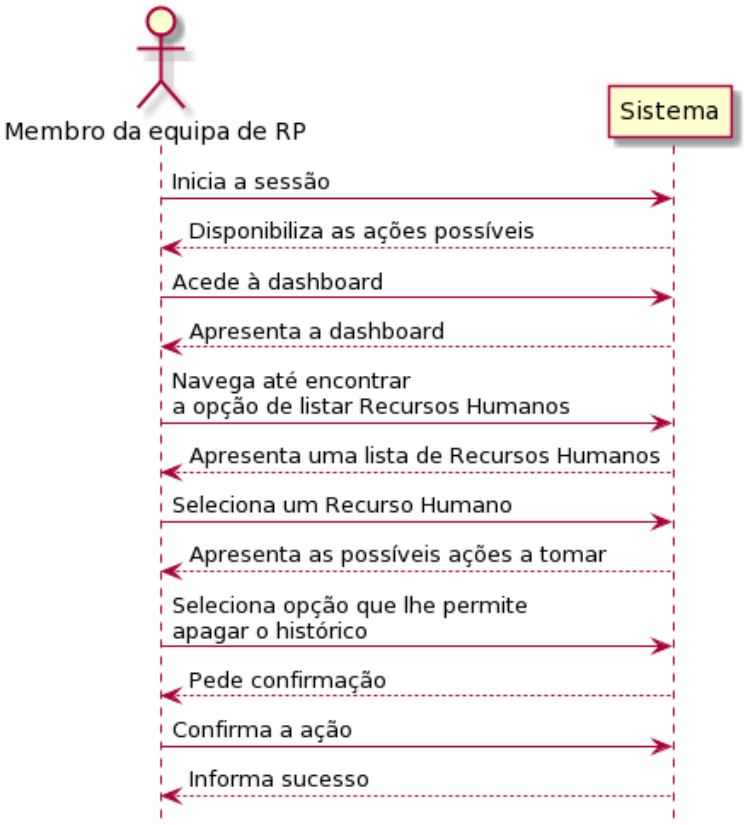

Figura B.17: SSD FR-17

B.18 FR-18 - Login

Tabela B.18: Detalhes do requisito FR-15

Descrição	Um utilizador inicia uma sessão no sistema.
Prioridade	Alta
Simulação/Sequência de Respostas	**Pré-condições:** • O sistema está ligado com o servidor LDAP da empresa. **Fluxo de Eventos:** 1. O utilizador acede ao sistema. 2. O sistema requisita ao utilizador o nome de utilizador e palavra-passe. 3. O utilizador introduz o nome de utilizador e a palavra-passe. 4. O sistema informa o sucesso da operação ao utilizador.

Pós-condições:

- O utilizador fica com uma sessão iniciada no sistema.

Validações:

- n/a

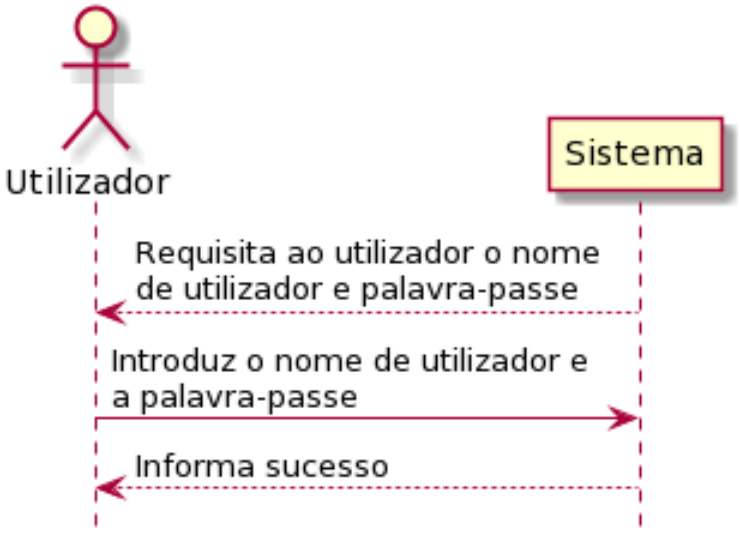

Figura B.18: SSD FR-18

Apêndice C

Indicadores Estatíticos

C.0.1 Total macro issues planned

Total macro issues planned: Representa a estimativa total, em horas, para o final do desenvolvimento de uma *release*.

```
1  FUNCTION TOTAL_MACRO(I_PRODUCT          IN VERSION.PRODUCT%TYPE,
2                       I_PRODUCT_VERSION IN VERSION.PRODUCT_VERSION%TYPE)
3      RETURN NUMBER IS
4      O_TOTAL_MACRO    NUMBER
5    BEGIN
6      SELECT SUM(ESTIMATION)
7        into O_TOTAL_MACRO
8        FROM ISSUE
9       WHERE ISSUE.PRODUCT = I_PRODUCT
10        and ISSUE.PRODUCT_VERSION = I_PRODUCT_VERSION;
11     RETURN O_TOTAL_MACRO;
12   END TOTAL_MACRO;
```

Listing C.1: Função TOTAL_MACRO

C.0.2 Total macro issues planned by team and by layer

Total macro issues planned by team and by layer; Representa a estimativa total, por equipa e por camada, em horas para o final do desenvolvimento de uma *release*;

```
1  FUNCTION TOTAL_TEAM_MACRO_PER_LAYER(I_TEAM                IN TEAM.NAME%TYPE,
2                                      I_PRODUCT              IN VERSION.
       PRODUCT%TYPE,
3                                      I_PRODUCT_VERSION IN VERSION.
       PRODUCT_VERSION%TYPE,
4                                      O_MACRO_PER_LAYER OUT
       T_LAYER_MACRO)
5       RETURN BOOLEAN IS
6    BEGIN
7       OPEN O_MACRO_PER_LAYER FOR
8          SELECT TASK.LAYER, SUM(TASK.ESTIMATION) AS MACRO
9          FROM TEAM, ISSUE, COMPONENT, TASK
10         WHERE ISSUE.PRODUCT = I_PRODUCT
11         AND ISSUE.PRODUCT_VERSION = I_PRODUCT_VERSION
12         AND TASK.PARENT_KEY = ISSUE.ISSUE_KEY
13         AND COMPONENT.NAME = TASK.COMPONENT
14         AND TEAM.ID_TEAM = COMPONENT.ID_TEAM
15         AND TEAM.NAME = I_TEAM
16         GROUP BY TASK.LAYER;
17      RETURN TRUE;
18   END TOTAL_TEAM_MACRO_PER_LAYER;
```

Listing C.2: Função TOTAL_TEAM_MACRO_PER_LAYER

C.0.3 Total development days

Total development days: Representa a quantidade de dias úteis planeados para o desenvolvimento daquela *release*.

```
1  FUNCTION TOTAL_DEV_DAYS(I_PRODUCT              IN VERSION.PRODUCT%TYPE,
2                          I_PRODUCT_VERSION IN VERSION.PRODUCT_VERSION%
       TYPE) RETURN BOOLEAN IS
3       DEADLINE_PLANNING     VERSION.DEADLINE_PLANNING%TYPE;
4       DEADLINE_DEVELOPMENT VERSION.DEADLINE_DEVELOPMENT%TYPE;
5       O_TOTAL_DEV_DAYS      NUMBER;
6    BEGIN
7       SELECT DEADLINE_PLANNING, DEADLINE_DEVELOPMENT
8          INTO DEADLINE_PLANNING, DEADLINE_DEVELOPMENT
9          FROM VERSION
10        WHERE VERSION.PRODUCT = I_PRODUCT
11           AND VERSION.PRODUCT_VERSION = I_PRODUCT_VERSION;
12      DBMS_OUTPUT.PUT_LINE('DEADLINE_PLANNING: ' || DEADLINE_PLANNING);
13      DBMS_OUTPUT.PUT_LINE('DEADLINE_DEVLOPMENT: ' || DEADLINE_DEVELOPMENT
        );
14      O_TOTAL_DEV_DAYS := PK_UTIL.BUSINESS_DAYS_INTERVAL(
       DEADLINE_DEVELOPMENT,
15                                                        DEADLINE_PLANNING
        );
16      RETURN O_TOTAL_DEV_DAYS;
17   END TOTAL_DEV_DAYS;
```

Listing C.3: Função TOTAL_DEV_DAYS

C.0.4 Remaining development days

```
FUNCTION REMAINING_DEV_DAYS(I_PRODUCT IN VERSION.PRODUCT%TYPE,
                            I_PRODUCT_VERSION IN VERSION.
    PRODUCT_VERSION%TYPE) RETURN NUMBER IS
    CURRENT_DATE TIMESTAMP WITH LOCAL TIME ZONE;
    DEADLINE_DEVELOPMENT VERSION.DEADLINE_DEVELOPMENT%TYPE;
    REMAINING_DEV_DAYS NUMBER;
  BEGIN
    SELECT SYSTIMESTAMP INTO CURRENT_DATE FROM DUAL;
    SELECT DEADLINE_DEVELOPMENT
      INTO DEADLINE_DEVELOPMENT
      FROM VERSION
     WHERE VERSION.PRODUCT = I_PRODUCT
       AND VERSION.PRODUCT_VERSION = I_PRODUCT_VERSION;

    IF (CURRENT_DATE > DEADLINE_DEVELOPMENT) THEN
      REMAINING_DEV_DAYS := 0;
    ELSE
      REMAINING_DEV_DAYS := PK_UTIL.BUSINESS_DAYS_INTERVAL(CURRENT_DATE,
        DEADLINE_DEVELOPMENT);
    END IF;
    RETURN REMAINING_DEV_DAYS;
  END REMAINING_DEV_DAYS;
```

Listing C.4: Função REMAINING_DEV_DAYS

C.0.5 Total remaining macro issues planned

Total remaining macro issues planned: Representa o total, em horas, do que falta desenvolver para concluir o que foi planeado para a *release*;

```
FUNCTION TOTAL_REMAINING_MACRO(I_PRODUCT IN VERSION.PRODUCT%TYPE
    ,
                               I_PRODUCT_VERSION IN VERSION.
    PRODUCT_VERSION%TYPE)
    RETURN ISSUE.ESTIMATION%TYPE IS
    O_TOTAL_MACRO ISSUE.ESTIMATION%TYPE;
  BEGIN
    SELECT SUM(ESTIMATION)
      INTO O_TOTAL_MACRO
      FROM ISSUE
     WHERE ISSUE.PROGRESS < 1.0
       AND ISSUE.PRODUCT = I_PRODUCT
       AND ISSUE.PRODUCT_VERSION = I_PRODUCT_VERSION;
    RETURN O_TOTAL_MACRO;
  END TOTAL_REMAINING_MACRO;
```

Listing C.5: Função TOTAL_REMAINING_MACRO

C.0.6 Days progress

Days progress: Representa o progresso dos dias a partir do momento em que começou o desenvolvimento da *release*;

```
FUNCTION DAYS_PROGRESS(I_PRODUCT            IN VERSION.PRODUCT%TYPE,
                       I_PRODUCT_VERSION IN VERSION.PRODUCT_VERSION%TYPE
    )
   RETURN NUMBER IS
   O_TOTAL_DEV_DAYS NUMBER;
   O_REMAINING_DEV_DAYS NUMBER;
 BEGIN
  O_TOTAL_DEV_DAYS := TOTAL_DEV_DAYS(I_PRODUCT => I_PRODUCT,
                                     I_PRODUCT_VERSION =>
    I_PRODUCT_VERSION);
  O_REMAINING_DEV_DAYS := O_REMAINING_DEV_DAYS(I_PRODUCT => I_PRODUCT,
                       I_PRODUCT_VERSION => I_PRODUCT_VERSION);
  RETURN ((O_TOTAL_DEV_DAYS - O_REMAINING_DEV_DAYS)/O_TOTAL_DEV_DAYS)*
    100;
 END DAYS_PROGRESS;
```

Listing C.6: Função DAYS_PROGRESS

C.0.7 Effort progress

Effort progress: Representa o progresso relativamente às horas que foram aplicadas no desenvolvimento da *release*;

```
FUNCTION EFFORT_PROGRESS(I_PRODUCT            IN VERSION.PRODUCT%TYPE,
                         I_PRODUCT_VERSION IN VERSION.PRODUCT_VERSION%
    TYPE) RETURN NUMBER IS
  O_EFFORT_PROGRESS NUMBER;
  BEGIN
    SELECT SUM(PROGRESS)/COUNT(*) * 100 INTO O_EFFORT_PROGRESS
    FROM ISSUE
    WHERE ISSUE.PRODUCT = I_PRODUCT
    AND ISSUE.PRODUCT_VERSION = I_PRODUCT_VERSION;
    RETURN O_EFFORT_PROGRESS;
  END EFFORT_PROGRESS;
```

Listing C.7: Função EFFORT_PROGRESS

C.0.8 Percentage of issues baseline without macro

Percentage of issues baseline without macro: Representa a percentagem de *issues* que pertecem à *baseline* da *release* mas que ainda não foram estimados pelas equipas de desenvolvimento.

```
1  FUNCTION PERC_ISSUES_BASELINE_WITHOUT_MACRO(I_PRODUCT IN VERSION
       .PRODUCT%TYPE,
2                                              I_PRODUCT_VERSION IN
       VERSION.PRODUCT_VERSION%TYPE)
3      RETURN NUMBER IS
4      WITHOUT_MACRO NUMBER;
5      TOTAL_ISSUES  NUMBER;
6    BEGIN
7      WITHOUT_MACRO := ISSUES_BASELINE_WITHOUT_MACRO(I_PRODUCT,
8                                                     I_PRODUCT_VERSION);
9      SELECT COUNT(*) INTO TOTAL_ISSUES
10     FROM ISSUE_BASELINE
11     WHERE ISSUE_BASELINE.PRODUCT = I_PRODUCT
12     AND ISSUE_BASELINE.PRODUCT_VERSION = I_PRODUCT_VERSION;
13     RETURN (WITHOUT_MACRO / TOTAL_ISSUES) * 100;
14   END PERC_ISSUES_BASELINE_WITHOUT_MACRO;
```

Listing C.8: Função PERC_ISSUES_BASELINE_WITHOUT_MACRO

C.0.9 Percentage of issues planned without macro

Percentage of issues planned without macro: Representa a percentagem de *issues* que foram planeados para a *release* mas que ainda não foram estimados pelas equipas de desenvolvimento.

```
1  FUNCTION PERC_ISSUES_WITHOUT_MACRO(I_PRODUCT IN VERSION.PRODUCT%
       TYPE,
2                                     I_PRODUCT_VERSION IN VERSION.
       PRODUCT_VERSION%TYPE)
3      RETURN NUMBER IS
4      WITHOUT_MACRO NUMBER;
5      TOTAL_ISSUES  NUMBER;
6    BEGIN
7      WITHOUT_MACRO := ISSUES_WITHOUT_MACRO(I_PRODUCT, I_PRODUCT_VERSION);
8      SELECT COUNT(*) INTO TOTAL_ISSUES
9      FROM ISSUE WHERE ISSUE.PRODUCT = I_PRODUCT
10     AND ISSUE.PRODUCT_VERSION = I_PRODUCT_VERSION;
11     RETURN (WITHOUT_MACRO / TOTAL_ISSUES) * 100;
12   END PERC_ISSUES_WITHOUT_MACRO;
```

Listing C.9: Função PERC_ISSUES_WITHOUT_MACRO