INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA INFORMÁTICA

Issue Creator

CÉSAR ANDRÉ DA ROCHA SEABRA

Outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Issue Creator

César André da Rocha Seabra

A dissertation submitted in partial fulfillment of
the requirements for the degree of Master of Science,
Specialisation Area of Software Engineering

Supervisor: Dr. Paulo Gandra
Company Internship Supervisor: Ivo Pereira
Company Technical Supervisor: Vítor Tavares

Porto, October 15, 2022

Dedicatory

Dedicated to all those who directly or indirectly helped me in this long academic journey, especially my family for all their support.

Thank you very much!

Abstract

In the constant development of the technological industry, when it comes to product development in terms of software development new trends tend to motivate the evolution of the software through the analysis of user feedback from issue tracking systems. This is because the ultimate success of any software and, as consequence, for any technology-driven company, falls on whether or not the developed solutions manage to fulfill the expectation of the final users.

E-goi is a company that provides a platform for multi-channel marketing automation that allows the integration of multiple channels from SMS and voice messages to e-mail and webpush.

When it comes to SaaS companies such as E-goi, user feedback becomes of extreme importance in order to improve its products and create value for both the user and the company. When managing user feedback, it is often important how it will be delivered to the development teams in such a way that the problem at hand becomes easily understood with the maximum information possible, to be able to replicate the bugs and to create new features for the product. This, of course, must be achieved with minimal impact when it comes to the analysis of the issues and consequent development.

However, the gathering and consequent delivery of this feedback to the product development teams, in E-goi, can come with some problems in both information standardization and duplicate prevention and extra costs when generated by the used tools when pursuing the objective of allowing the entire company to provide said feedback as well.

So, to solve this problem, E-goi decided to create a tool that allows all the collaborators to submit issues - the Issue Creator. Nevertheless, other described problems still need to be solved. Here, is where this project comes into play by developing a revamp of this platform and enabling the creation of standardized issue reports, issue duplication prevention, and the implementation of other features that involve the integration of different platforms to simplify the actions that are essential to the product development teams.

In this report, an introduction to the identified problem is described, along with the objectives and methodology followed. After this, a full contextualization on how the E-goi organizational departments are distributed, with an emphasis on the product development department, and their processes in software development. Subsequently, an analysis of the value of the solution and the requirements gathered through the elicitation phase as part of the requirements engineering practice is made, passing by a detailed view of the proposed design to develop the platform. Finally, the developed platform was evaluated both from the technical aspect through tests and quality aspects comprehended by the users, by taking advantage of the stakeholder answers gathered from inquiries performed.

Keywords: Software Development, Agile Methodologies, BPMN, Bug Report, Issue Report, Issue Duplication, System Integration

Resumo

Com o constante desenvolvimento da indústria tecnológica, quando se trata de desenvolvimento de produtos, mais concretamente em termos de desenvolvimento de *software*, as novas tendências tendem a motivar a evolução do *software* através da análise do *feedback* dos clientes reunido nos sistemas de gestão de tarefas. Isto deve-se essencialmente ao facto do sucesso de qualquer *software* e de qualquer empresa tecnológica, depender do facto das soluções desenvolvidas conseguirem ou não atender às expectativas dos utilizadores finais.

A E-goi é uma empresa que disponibiliza uma plataforma de automação de marketing multicanal que permite a integração de múltiplos canais desde SMS e mensagens de voz a e-mail e *webpush*. No que concerne empresas SaaS como a E-goi, o *feedback* dos utilizadores torna-se de extrema importância para melhorar os seus produtos e criar valor tanto para o utilizador como para a empresa. Ao gerir o *feedback* do utilizador, muitas vezes é importante como ele será entregue às equipas de desenvolvimento, de forma a que o problema em questão seja facilmente entendido com o máximo de informações possível, aquando se tenta reproduzir os *bugs* reportados ou mesmo desenvolver novas funcionalidades. Isso, é claro, deve ser alcançado com o mínimo de impacto aquando a análise das *issues* e consequente desenvolvimento.

No entanto, a recolha e consequente entrega deste *feedback* às equipas de desenvolvimento de produto, no E-goi, pode acarretar alguns problemas quer na uniformização da informação, quer na prevenção de duplicações e custos extra gerados pela utilização das ferramentas de gestão de *issues* pelos diversos colaboradores da empresa, de forma a permitir que estes também possam reportar novos problemas.

Assim, para resolver esta problemática, a E-goi decidiu criar uma ferramenta que permitisse a todos os colaboradores submeterem novas *issues* - o Issue Creator. No entanto, outros problemas descritos ficaram por resolver. É aqui que entra este projeto, ao desenvolver uma reformulação desta plataforma e permitir a criação de relatórios de *issues* standerdizados, prevenção de duplicação de relatórios e a implementação de outras funcionalidades que envolvem a integração de diferentes plataformas, de forma a simplificar as ações que são essenciais para as equipes de desenvolvimento de produtos.

Neste relatório é descrita uma introdução ao problema identificado, bem como os objetivos e a metodologia seguida. A seguir, é feita uma contextualização completa de como estão distribuídos os departamentos da E-goi, com ênfase no departamento de desenvolvimento de produto e nos seus processos no desenvolvimento de *software*. Posteriormente, é feita uma análise de valor da solução e dos requisitos levantados na fase de elicitação como parte da prática de engenharia de requisitos, passando por uma visão detalhada do *design* proposto para o desenvolvimento da plataforma. Finalmente, a plataforma desenvolvida é avaliada tanto do aspeto técnico por meio de testes quanto dos aspetos de qualidade compreendidos pelos utilizadores, através da analise das respostas obtidas pela realização de questionários.

Acknowledgement

Firstly, I would like to thank Professor Paulo Gandra for his advice and support, which allowed me to complete this project successfully.

I also thank E-goi for the opportunity and support for this thesis, including all the employees for their friendliness and fantastic personality.

Finally, I would like to thank Vítor Tavares for his follow-up and humor and, last but not least, a big thank you to Ivo Pereira for his patience and incessant support in carrying out the thesis.

Contents

List of Figures

List of Tables

List of Acronyms

AHP	Analytic Hierarchy Process.
AI	Artificial Intelligence.
API	Application Programming Interface.
BDD	Behavior Driven Development.
BPM	Business Process Management.
BPMN	Business Process Model and Notation.
CEO	Chief Executive Officer.
CI	Consistency Index.
CR	Consistency Ratio.
CTO	Chief Technology Officer.
FEE	Fuzzy Front End.
GRASP	General Responsibility Assignment Software Patterns.
HD	Head of Department.
HTML	HyperText Markup Language.
ITS	Issue Tracking System.
JIT	Just In Time.
JQL	Jira Query Language.
KPI	Key Performance Indicator.

NCD	New Concept Development.
NPD	New Product Development.
OKR	Objective Key Result.
PM	Product Manager.
PT	Product Team.
QA	Quality Assurance.
QEF	Quantitative Evaluation Framework.
RAM	Random Access Memory.
REST	Representational State Transfer.
SaaS	Software-as-a-Service.
SRE	Site Reliability Engineering.
SUS	System Usability Scale.
SWOT	Strengths, Weaknesses, Opportunities and Threats.
TDD	Test Driven Development.
TL	Tech Lead.
UI	User Interfce.
UML	Unified Modeling Language.
UXA	User Experience Assurance.
WIP	Work In Progress.

Chapter 1

Introduction

This chapter focuses on the presentation of the project theme. Among the topics discussed are the context of the problem, the presentation of the company in which the project took place, the motivation for its development, the objectives, the methodology used, and finally, a discussion of the document's structure.

1.1 Context and Company Presentation

The current project was developed at E-goi, founded in 2008 and located in Porto, Matosinhos. It is considered a medium sized business that has more than 100 employees and serves more than 450 thousand clients world-wide, with the majority of the market share in Brazil.

E-goi's main business consists in the availability of a Software-as-a-Service (SaaS), providing a multi-channel marketing automation platform that is able to integrate diverse communication channels such as Smart SMS, voice messages, push notifications, webpush and the most used ones - e-mail and SMS.

The concept behind a SaaS is that instead of a customer buying a software license and having it run in an environment created for them, they pay a periodic subscription, where the necessary resources, the running environment and data center are all managed and provided by the vendor, which is also responsible for the maintenance, support activities [1].

In the case of E-goi, its platform allows its clients to manage all the contacts of their customers and to track and engage the behavior of said customers in order to act upon, by automating and simplifying all the process of creating, sending and tracking marketing campaigns and generating analytic reports from these.

Therefore, the company's main mission is to develop digital communication solutions, accessible to everyone, especially to small and medium businesses. This is done in a way that helps their clients to better relate to their customers, thereby creating more value for everyone involved.

Currently, more and more companies are approaching a dynamic of delivering cloud-based products, more specifically in the form of Software-as-a-Service (SaaS), in order to encompass the constant evolution of the surrounding market.

As a result, other direct competitors are trying to adapt their business to their surroundings, as is also the case with E-goi. In order to increase business value, the company must keep a close relationship with not only the newest technological opportunities but also the needs and feedback of their customers. This relationship offers new challenges and opportunities to improve both the software development process and customer relationship.

1.2 Motivation

As previously stated, with the increasingly faster growth of the company and its product (since each day nearly 350 new accounts are created) it becomes essential that the development process and all other inherent processes must be revisited and rethought in order to increase their performance and efficiency.

In recent years, E-goi has focused on increasing the number of customers and investing more in improving their main product, creating new features and developing custom solutions for their clients. This led to an increase in the complexity of the product, which, in turn, resulted in the need for an increasing investment in human resources.

Since the issue management inside E-goi is made trough Jira[1], the fact that the creation of new users in this platform brings additional costs per user, it was not financially viable to support the interaction of all the company collaborators with the platform, which didn't allow them to directly report any issues to the development team, task that was a must have for the work organization.

This problem led to the creation of the platform **Issue Creator**, that communicated directly with the Jira platform and allowed the submission of issues trough Representational State Transfer (REST) requests. This platform was a simple HyperText Markup Language (HTML) page with a form that stakeholders used to submit issues.

This, however, led to other issues affecting the development team. Since the form was unique and static, the creation of the issues was the same for all their types (bug, improvement, feature, etc). This impacted the efficiency of the development process since the Product Managers needed more time to clearly identify what each issue corresponded to. Additionally, the absence of rule enforcement led to issues without proper standards, lacking information and with ambiguous interpretation.

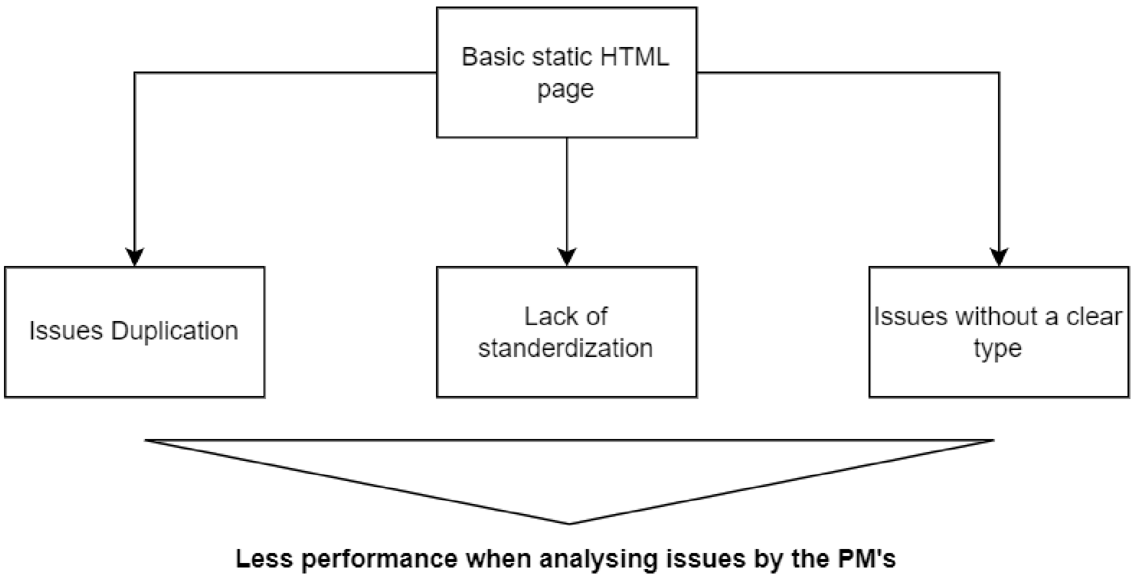

Basic static HTML page
Issues Duplication
Lack of standerdization
Issues without a clear type
Less performance when analysing issues by the PM's

Figure 1.1: Identified problem general overview

[1]https://www.atlassian.com/software/jira

Another of the problems consisted in the increase of duplicate issue reported and the lack of integration possibilities with other platforms such as the ones used by the customer success department (LiveAgent[2]) and the platform used by the entire company for process management - Bitrix24[3]. Figure 1.1, represents an overview of the problem identified.

Since the company's product depends primarily on the development process, it becomes crucial for the organization that the issue creation process is clear, fast, and normalized in order to increase efficiency.

This raised the opportunity to perform the revamp of this platform in order to improve several aspects such as: platform integrations, issue duplication reduction, issue reporting normalization and performance (e.g. by allowing the addition of values that affect the issue prioritization), and improvements to the client's issuer feedback process.

1.3 Objectives

Upon analyzing the problem and retrieving the needs of the stakeholders, it became apparent that it was necessary to restructure the application in order to create an integrated solution for the submission of requests of new bugs and features in order to increase the performance of the development team when analyzing and classifying issues. This application should be able to increase the quality and quantity of the information reported in the issues created, diminishing of duplicate reports, distinction of issue types, provide visibility and a work priority for the entire company and, finally, the integration with other systems.

In order to achieve this objective, the following tasks were identified as essential to be able to construct the final solution:

- Identify and analyze the process of software development inside the company and perform informal interviews in order to gather all the requirements to be applied to the platform;
- Identify and explore already conducted studies and information that may allow projecting the solution for the development of the pretended solution and, at the same time, explore the technologies that will be covered;
- Define the solutions to the presented problems as well as an architecture for the solution envisioning a scalable and robust solution;
- Implementation of the proposed solution and posterior validation of its performance in the company's problem resolution.

1.4 Methodology

This project firstly started with the research on the areas that where most relevant in order to find solutions to the presented problems. In this phase, information was gathered and summarized that allowed for the development of a theoretical background and provided directions for the solution of the problem.

Then a qualitative and quantitative study was conducted in order to identify the processes inherent to the development process and all cross-functional processes that occur when

[2]https://www.liveagent.com
[3]https://www.bitrix24.com

dealing with customer issues across multiple departments, as well as which problems were more apparent in these processes. This study was conducted through informal and formal surveys and observation.

Using the gathered data, it became possible to assemble a comprehensive picture of how these processes work. This picture helped us identify the requirements that could be implemented in the final solution to fill the gaps experienced by the stakeholders. This allowed one to then proceed to the analysis of both the software and business value of the solution.

The piece of software was designed and implemented in order to integrate all the tools used by the company. It also allowed to implement all the requirements gathered in the previous step, using the appropriate practices of Software Engineering.

Finally, software tests and quality assurance methodologies were applied to the project in order to ensure the final solution was in accordance with the planning, and that it met all the requirements specified throughout the analysis and design phases.

1.5 Document Structure

This document is divided into eighth chapters. Presented in this chapter is a general context of the problem. It also includes the company's presentation, the reasons for the development of the solution, the objectives that must be met and the methodology to be used.

The second chapter documents all the processes used by the company for the development of their product. This gives general context on the flow of information, decisions and processes.

The third chapter discusses the theoretical background on the topics involved in the development of this project, such as tasks prioritizing in issues, the correlation between SaaS and customer feedback, Business Process Management (BPM), software development methodologies and tools for issue management.

In the fourth chapter, the analysis of the problem is presented, as well as the proposed solutions for the difficulties identified in the previous chapter and, finally, the analysis of the value of the project to the company.

In the fifth and sixth chapters are presented the design of the solution for the software developed and the implementation process and overview, respectively.

The seventh chapter is used to describe the tests implemented in order to perceive if the final solution has all the elements to consider itself a valid project.

Last but not least, in the last chapter, are presented the conclusions from the developed project, along with some suggestions for the future work development.

Chapter 2

Context

In order to contextualize the problem and to perceive the reality of the software development processes inside the company, an analysis of these processes must be made in order to verify how they are made. In addition to the product department, these processes may involve other departments across the company.

This chapter contains all the information relative to the company structure, with special attention to the product development team and their processes in the releasing of newly developed software. As such, this chapter is devoted to discussing the following topics: description of company structure with focus on the product development department, operational responsibilities inside the product department, customer feedback gathering process and tools, the software development process inside the department described using Business Process Model and Notation (BPMN) (described in the Section 3.5 of Chapter 3), and, finally, the software development methodology.

2.1 Company Structure

As in any other medium-size company, E-goi is segregated into multiple departments, each one with a well-defined set of processes and responsibilities. Figure 2.1, presents the overall view of the different departments present in E-goi.

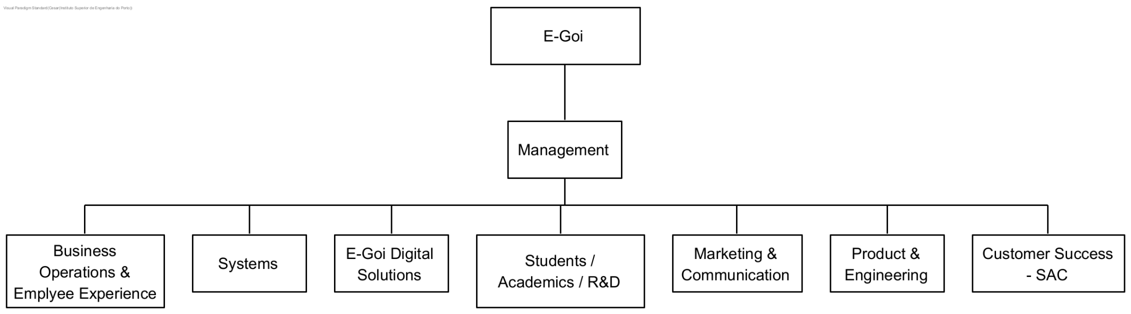

E-Goi
Management
Business Operations & Emplyee Experience
Systems
E-Goi Digital Solutions
Students / Academics / R&D
Marketing & Communication
Product & Engineering
Customer Success - SAC

Figure 2.1: E-goi's company structure

All departments directly or indirectly affect the development of the product within the company. However, the ones with the most impact are the departments of customer success and the department of E-goi digital Solutions, since they are the ones that report the higher amount of issues.

2.1.1 Customer Success Department

The Customer Success department is responsible for increasing customer satisfaction by providing both technical and general support to customers. Their main objective is the reduction of client churn by providing solutions to their problems and gathering their feedback in order to retain them.

The product support feature is provided to all types of clients, even though the personalized technical support is only available to paying E-goi accounts. Accounts with free access have access to automated support, courses, and helpdesk.

The way that the SAC department gathers and organizes the client's needs is through the tool LiveAgent. This tool allows them to collect issues in the form of tickets. More on this feedback process is present in the Subsection 2.3

2.1.2 E-goi Digital Solutions Department

The E-goi Digital Solutions department is one of the most significant departments inside the company in terms of revenue since they correspond to 70% of the annual revenue of the company by managing more than 400 corporate accounts.

In addition to gathering new corporate clients, they also create custom marketing plans, guide the new clients through the E-goi platform experience, and provide regular feedback gathering and support to their clients.

They are divided into three sub-departments, namely:

- Account: which is responsible for the management of the corporate customer portfolio;
- New Business: responsible for acquiring prospects for corporate clients;
- Agency: responsible for the development of new corporate products and made-to-measure projects

2.1.3 Product and Engineering Department

On the other hand, we have the product & engineering department whose main focus is the development of products that offer solutions to the user's problems, bringing value to their business and contributing to meeting the company's objectives.

This department is divided into three layers. Each layer comprises multiple product teams with the same drive, coordinating with each other, focused on very similar Key Performance Indicator (KPI) and goals. On the other hand, a product team corresponds to a multidisciplinary team responsible for specific products or components of the ecosystem. They are autonomous, self-organized, and self-managed and each one includes a manager who defines the vision, the strategy, and the operations.

In Figure 2.2 is shown an overview of the product department structure, which shows the different layers.

As stated in the previous figure, there are three different layers: core, backbone and business. Each consists of a set of KPI and objectives and aggregates a group of well-defined teams. Each one of them is explained in more detail in the topics below.

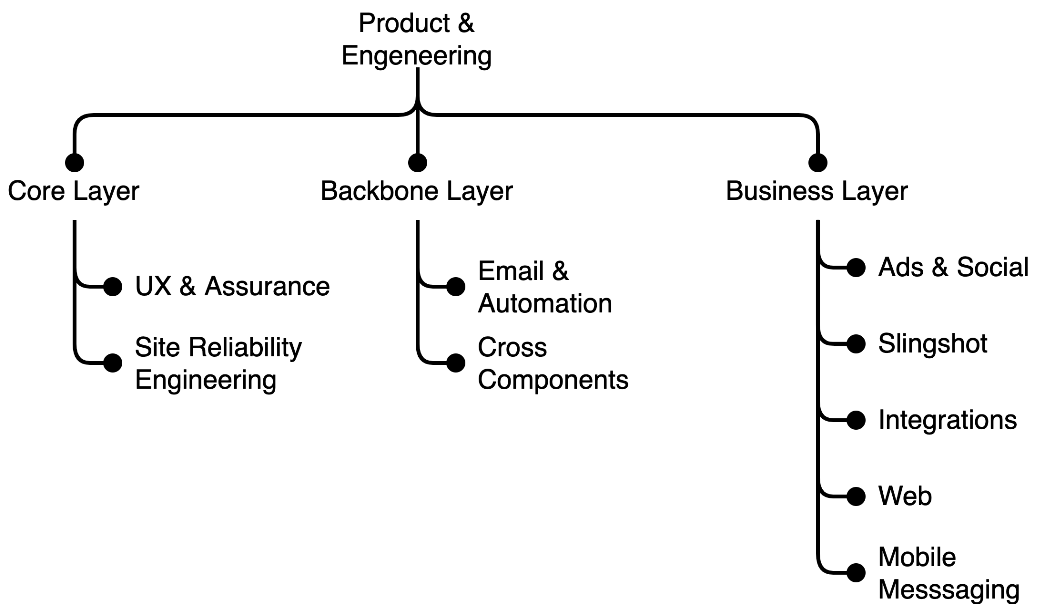


Figure 2.2: Product & Engineering department structure

Core Layer

The Core layer comprises a technology-centered team (Site Reliability Engineering (SRE)), whose central focus is the improvement and maintenance of the technology and the architecture of the ecosystem by refactoring code, switching to a better framework, and, sometimes, researching on how to invest in new technologies. Their main responsibilities include software architecture, data architecture, release management, and code review. Their KPI are based on performance, scalability, and stability of the overall ecosystem, such as response times and page loading times.

They also include a design team, which prioritizes the beauty and overall experience of the products, reducing the dead-end experience. Some of their main tasks involve onboarding experience, application content, knowledge base content, customer research, product design, and application testing.

Backbone Layer

The Email & Automation team is responsible for the management of the e-mail and messaging automation (Autobots software component). This component enables the automation of all the marketing cycle processes of the customers. By using this tool, E-goi customers can track their clients' behavior based on factors such as email openings, total clicks, profiles, etc. This allows the creation of a personalized marketing campaign for each client group in order to maintain client loyalty as high as possible. In terms of responsibilities, this team acts upon:

- Corrective maintenance;
- Evolutive maintenance;
- Development of E-goi's highest impact and highest usage components (Autobots).

Meanwhile, the Cross Components team focuses on reducing dead-end experiences and increasing business as well as the purchase and payment process. Their main focus is:

- Experience and conversion-focused components such as payment intents, checkout, products, etc;
- Components with a major focus on the user's initial experience such as contacts, lists, and segments.

Business Layer

The Business Layer consists of teams that are growth-centered, which means that its main goal is the growth of the company's business by improving and building new products.

The main KPIs of this layer are acquisition, conversion, engagement, and retention, as well as revenue. It consists of five teams: Ads & Social, Slingshot, Integrations, Web, and Mobile Messaging.

The Ads & Social team is responsible for the Ads component, which allows users to capture clients that are not yet aggregated in the database (contacts). In order to achieve this, the platform allows the creation of simple ads in the main search engines and social networks all at once. Aside from that, data can be tracked centrally, with all information gathered into reports that permit users to analyze and decide upon the success on various channels. The main responsibilities of this team are:

- Boost the email campaigns, landing pages, and products;
- Distribution to the various communication channels (Facebook, LinkedIn, Instagram, etc);
- Simplify the process of creation of reports and data analysis of campaigns launched in social networks.

On the other hand, the Slingshot team has the objective of increasing the number of transactional messages sent. Whenever an action is performed on the platform, these messages appear. The main products and objectives of this team are:

- Transactional Email, SMS and Push notifications;
- Verification and authentication;
- Link shortener;
- Multi-channel messaging;
- Registered messages (certified messages with evidence).

The Integrations team focuses on increasing integration value in the company ecosystem. The integrations allow the E-goi clients to synchronize the databases of their stores with E-goi and to integrate E-goi solutions within various frameworks and through their API. Thus, the main responsibilities and areas of this team are:

- Perform integration partnerships;
- Development of integration plugins (e.g. with WordPress, Prestashop, etc);
- Development and maintenance of their Application Programming Interface (API) (both V2 and V3).

In the case of the Web team, their primary focus is on enhancing the client's database and getting clients involved. This team works with tools that allow the user to capture new clients, create relevant content to convert visitors into buyers, and decrease the churn rate. The commitment and activities of this team are based on the development and maintenance of the following functionalities:

- Embed and Pop-up forms;
- Landing pages;
- Webpush notifications;
- Cart abandonment campaigns.

Finally, the Mobile Messaging team is responsible for ensuring the performance of operations related to mobile messaging. This includes all the following services:

- SMS and Smart SMS;
- Voice broadcasting and IVR (Interactive Voice Response);
- Mobile Push Notifications.

2.2 Operational Position Responsibilities

In order to better comprehend how the different tasks are distributed among all the stakeholders in the process of product development in E-goi, a table with all the roles and their respective responsibilities was delineated. This information can be observed in the following Table 2.1.

Table 2.1: Operational position responsibilities inside the product department

Role	**Description**
Chief Executive Officer (CEO)	It's the person responsible for creating the annual vision for the company and for defining the strategy and objectives according to the company mission, product, mission, and operational needs
Chief Technology Officer (CTO)	The person who is responsible for technological necessities, as well as their research and development. He is also in charge of defining the product vision, organizing the teams, and establishing the department culture
Tech Lead (TL)	The person responsible for helping reach the technical, implementation, and architectural decisions inside a team

Role	**Description**
Product Manager (PM)	Responsible for all the components in a business area, ensuring the product delivery in accordance with specifications. They are also responsible for managing all the aspects related to the team such as resources, people, and projects and providing guidance, envisioning the fulfillment of the objectives.
Product Team (PT)	Focused on the development of products that offer solutions to users and for corrective maintenance. They are also responsible, during the planning phase, for analyzing, along with their PM, the estimated effort of the tasks to perform.
Masters	They correspond to all the managers of the existing teams in the department of product development.
Head of Department (HD)	Responsible for gathering the requirements of the product. They correspond to the persons in charge of each department. These, along with the remaining members of the company, collaborate with the product department to report issues related to the system.
Quality Assurance (QA)	The team is responsible for assuring that the product is according to the specifications and the required quality by testing the new corrections and functionalities.
User Experience Assurance (UXA)	Team responsible for assuring the highest possible user experience. This involves the design of all the processes of acquisition and integration of the product, including branding issues, design, and usability. This team is also responsible for the content creation for internal and external communication, including the maintenance of the knowledge base.

2.3 Feedback Gathering Process

As with any other technology company and especially those that provide SaaS models, they always seek to increase the amount of retained customers, so they need to monitor the constant feedback provided by the clients, as it is extremely important in order to keep the improvement flow on the products provided.

In E-goi, this feedback can be obtained from various sources, such as Google Analytics, Full Story, forms, ChatGoi (live chat), the E-goi community, and LiveAgent. All this information coming from different sources is analyzed and filtered in order to extract and define the information that can lead to the creation of new issues - may it be bugs, features, or improvements - that are later used by the product team in order to define the road maps of action.

In E-goi, there are numerous feedback input systems and the information recorded is not stored in one place, but distributed among several tools and systems, which forces the product management team to invest considerable resources (especially time) in gathering and filtering feedback from numerous sources, leading not only to duplication of data, but also (to some degree) to feedback starvation, or unused data. These fonts of information are listed below.

Chatgoi

A chatbot, available on the E-goi main page and through the "Help" button, allows the user to interact with and receive information from the knowledge base. In the end, the userAs a result, the user is faced with the question of whether the information presented is relevant or not, and if not, the user can contact customer support.

The data collected in this system is only used for statistical purposes in order to compare the number of usages and types of interaction with the platform.

Issues in Jira

The platform is used by all employees of the company to create tasks for the product, development, and systems teams. The issue submission takes place through E-goi's internal admin area (The Issue creator) which is integrated directly with Jira, which is the tool used by the three teams to manage and organize the tasks of the product.

With the help of Jira API integration as well as components that keep track of the team lead in each of the sub-areas of each team, tasks are automatically delegated to the person responsible for that area and the information is stored directly in Jira database.

The evaluation of this type of feedback is made directly on the platform. This tool is used not only to obtain feedback from users but also to provide a response to the person that issued it and, subsequently, to the other teams, since each issue correlates to a determined stage (stage of the development flow that the issue is in) which can provide insights about the current situation with that issue.

Helpdesk - LiveAgent

Used by the customer support team, it corresponds to an integrated online customer service system, where the main functionality is to capture user interactions whenever they request support both on the "Help" page and in the ChatGoi feature.

User interactions are handled through tickets, which contain tags indicating the area (sales, technical, etc) and priority of the problem. Currently, the product team analyzes and evaluates the various tickets that are created and takes action whenever they find a point for improvement.

E-goi Community

Refers to a form that is shown to the users of the platform where they have the possibility to submit suggestions for improvement. The data generated by this method is saved in the same area as the raw data collected by ChatGoi.

The information produced by this method is directly evaluated by the product manager, who promptly responds and determines the status of the said suggestion in four categories: rejected, planned, evaluated, or implemented.

Google Analytics and Full Story

These two tools are used mainly by the product team in order to identify possible points for improvement. These tools work essentially by recording user interaction with the E-goi platform allowing a careful and more precise identification of bugs and improvements.

It is relevant to mention that this type of analysis data extraction is mainly through manual observation. In this type of analysis, if any issue is found it is directly opened on the Jira platform and delegated to the respective component.

2.4 Product Development Process

The definition of the roadmap of the processes present in the company's product development was possible through the analysis of the provided documentation, the formal interviews with the CTO and other product managers and through field observations of the daily activities of the team members. This general roadmap is presented in the following Figure 2.3.

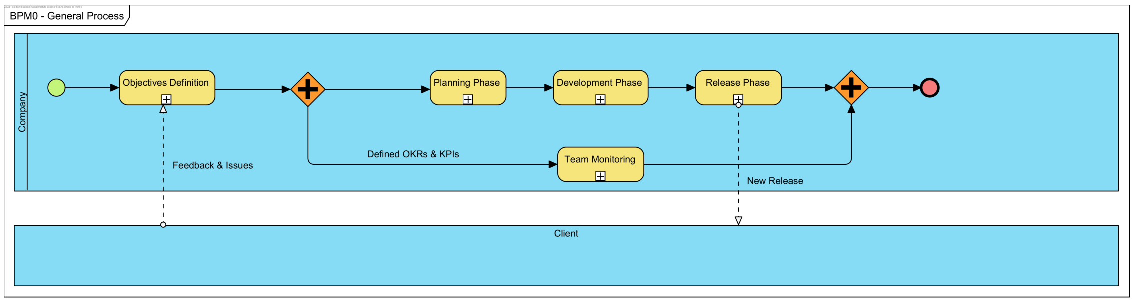


Figure 2.3: Product development process general view

According to the figure, there is a first phase of defining objectives and then there are two flows of parallel processes - a sequential flow with three processes referring to software development more specifically (planning, development and release) and another referring to the monitoring of the teams of product development.

The objective definition phase comprises the definition of the work plan that will be developed during the trimester, according to the strategy defined by the company (CEO). At this stage, according to the survey of the needs of each department, Objective Key Result (OKR) are defined for each team, so that the employees of each product department have a vision of the goals to be achieved. These needs are always aimed at improving the product, thus meeting the needs of customers, their problems (bugs), and suggestions (improvements, features, etc).

In the software development sequence, there are three phases. The first is the **Planning**, which is a process executed weekly, where it is discussed what was done in the previous sprint and what will be developed in the next sprint, the needs related to the integration of components, and the suggestion of improvements according to the conclusions drawn from the work presented during the week.

Then we have the **Implementation**, a process that comprises the development of software, according to the objectives established and the needs previously established and with the appropriate prioritization.

Finally, the **Release** process that consists of the integration of the work developed by the product teams and the subsequent launch of a newly developed version of the product. If the features developed are relevant, the release is then communicated internally and externally.

Regarding the phase parallel to development, we have team monitoring. In this process, transversal to the development processes, a monthly analysis of the work is done and unforeseen events are carried out as an adjustment of the defined objectives, if necessary. This analysis is performed based on OKR and KPI assigned to each development team.

Next, each of the phases presented above is detailed with the help of BPMN diagrams. In these processes, it is possible to verify the workflow and the interaction between the different actors according to the table present in the Section 2.2 in the Table 2.1.

2.4.1 Objectives Definition Process

As previously mentioned, the process of defining the objectives is carried out quarterly, and is based on the definition of what will be developed by the product teams. In the Figure 2.4, the tasks inherent to this process are described, as well as the actors responsible for each of them.

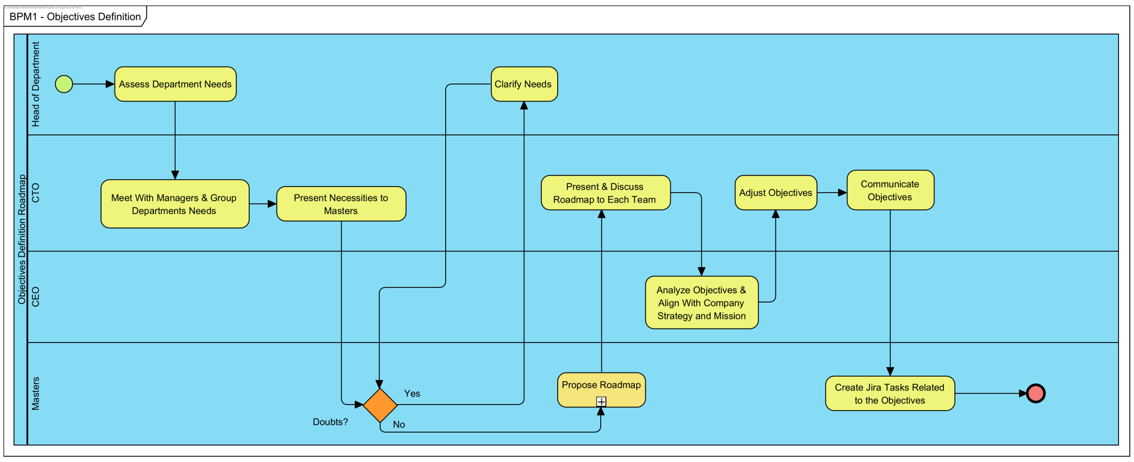

BPM1 - Objectives Definition
Objectives Definition Roadmap
Head of Department
CTO
CEO
Masters
Assess Department Needs
Clarify Needs
Meet With Managers & Group Departments Needs
Present Necessities to Masters
Present & Discuss Roadmap to Each Team
Adjust Objectives
Communicate Objectives
Analyze Objectives & Align With Company Strategy and Mission
Yes
Doubts?
No
Propose Roadmap
Create Jira Tasks Related to the Objectives

Figure 2.4: Objectives definition process in product development

The process begins with a survey of the needs of each department manager and then the CTO meets with the managers in order to aggregate all the needs. This is so that they can then present these to the other product managers. These needs are then analyzed by each master and decisions are made, so that they can create the quarter work roadmap, as shown in the figure above (Figure 2.4), with more detail in the Figure 2.5. The creation of this roadmap allows the masters to balance the objectives to be accomplished during the quarter and to meet the OKR, associated with the KPI.

From there, the CTO meets once again with each team in order to analyze and discuss the proposals in order to group and prioritize the most relevant ones. The result of this meeting is then presented to the CEO in order to align the objectives with the strategy outlined by the company and, according to this result, the CTO adjusts the objectives and communicates them to the masters.

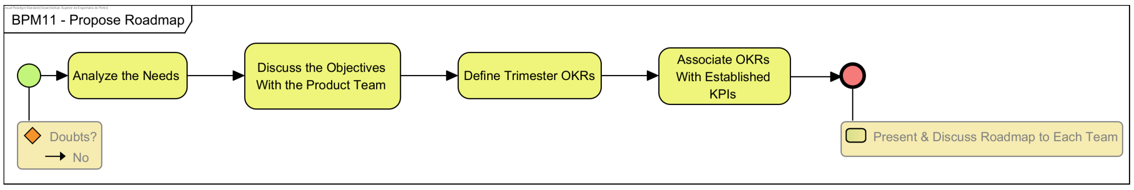

BPM11 - Propose Roadmap
Analyze the Needs
Discuss the Objectives With the Product Team
Define Trimester OKRs
Associate OKRs With Established KPIs
Doubts?
No
Present & Discuss Roadmap to Each Team

Figure 2.5: Propose roadmap sub-process

As each master becomes aware of these objectives, it is then possible to create and share tasks among team members by placing and explaining them in Jira, the platform used by the product development department for task tracking.

2.4.2 Planning Phase Process

As previously mentioned, the planning process is carried out weekly and is distinguished by the following steps: prioritization of needs, description of the previous sprint and definition of the next sprint. The prioritization of needs is the responsibility of the masters and is the first task to be done (as shown in the Figure 2.6).

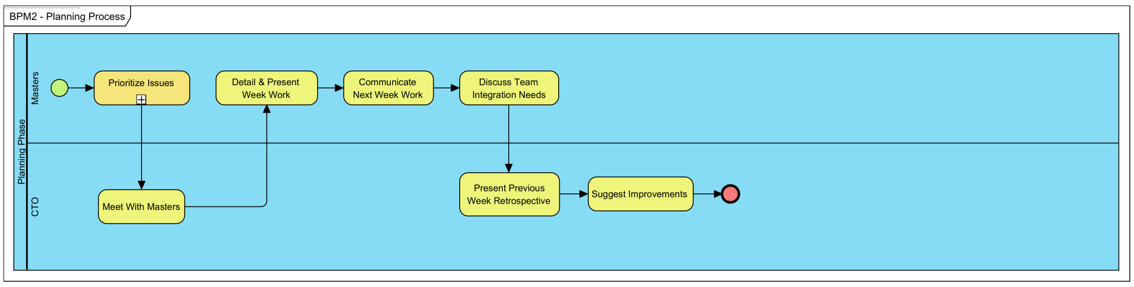

BPM2 - Planning Process
Planning Phase
Masters
CTO
Prioritize Issues
Detail & Present Week Work
Communicate Next Week Work
Discuss Team Integration Needs
Meet With Masters
Present Previous Week Retrospective
Suggest Improvements

Figure 2.6: Planning phase process

By analyzing the descriptions of the issues discussed by company employees, a process is in place to categorize their priority. This is so that only the most significant issues can be developed, and those that are in line with the objectives set in the previous phase. In the Figure A.1 in Appendix A, the prioritization sub-process is described in detail.

In this sub-process, the master analyzes the issues, changes their status to "In Analysis" and scores them. If it is a bug, it is examined and an attempt is made to replicate it. On the other hand, if it is another type of issue, the objectives and scope are defined and it is decided whether to implement it or not. If yes, then the issue is reported to the development phase, if not, then the appropriate resolution is assigned (as per Subsection 2.5.2, Table 2.3) in order to inform the issue creator that it has been analyzed and dealt with.

Regarding the issue of scoring, this is carried out both with a focus on the customer and on the product. This scoring was adopted in order for the issues created to have into account the importance of the issue, the type of client associated with it (the product plan that the customer has) and the degree of which the client has in terms of discomfort and the potential churn of such client.

According to the model used, in relation to the customer, there are three classification vectors that must be taken into account in order to correctly assess the priority of an issue.

These consist on three fronts which are enumerated bellow:

- **Customer Impact**: corresponds to the day-to-day impact for the customer (e.g. none, discomfort, blocker);
- **Customer Business Value**: correlates to the type of account of the customer, the segment in which the customer is inserted (e.g. trial, free, pro, corporate, etc);
- **Customer Satisfaction**: related to the relationship status of the customer with the platform (e.g. satisfied, neutral, lack of confidence, churn eminence, etc).

Regarding the product perspective, the scoring is calculated taking into account the following factors:

- **Product Impact**: the measure of how big the effect will be on the users. It takes into account the impact on features, the product, and the business;
- **Product Reach**: reflects the potential number of affected users. How many targets it will reach;
- **Product Effort**: the effort needed to develop the issue, the longer the time invested, the higher the cost, and, as such, the lower the priority.

After making the classifications in each of these vectors, the final score is calculated taking into account the formula $\frac{(clientperspective+productperspective)}{30} * 10$ and the MoSCoW method is applied according to the final result.

The MoSCoW (must have, should have, could have, won't have) method, is used to prioritize stories in iterative approaches. It provides the means to reach a common understanding of the relative importance of the delivery of a piece of value in the product [2]. As stated it comprises the following categories and appropriate scoring variations:

- Must-have (8-10): major issue without which the product does not work;
- Should-have (6-8): issue with significant business value, but the product can work without it, although it is important that it is included;
- Could-have (4-6): the issue does not have significant business value and can be left out, it can be applied in future versions depending on the effort required;
- Won't have (0-4): issue with a very low business value, so it is not considered for development.

After the priorities are defined, as well as the issues that will be developed, these are distributed among the teams. Also in this sequence, meetings are held with all the masters, in order to inform what was done in the previous sprint and what will be done in the next, as well as exchanges of information about possible integration processes between teams. Here is also carried a retrospective of problems that have occurred together with the CTO in order to be able to take the best measures to overcome them.

2.4.3 Development Phase Process

This phase corresponds to the development of the issues declared in the previous phase. The process starts by changing the status of the issues to "For Dev", in which it is verified whether it is a bug or not (as shown in the Figure 2.7). If it is not a bug, the flow begins with the creation of a mockup, which aims to create a general flow of the functionality/improvement to be carried out. This mockup then passes to the UXA team, which is responsible for its design. The issue then moves to development.

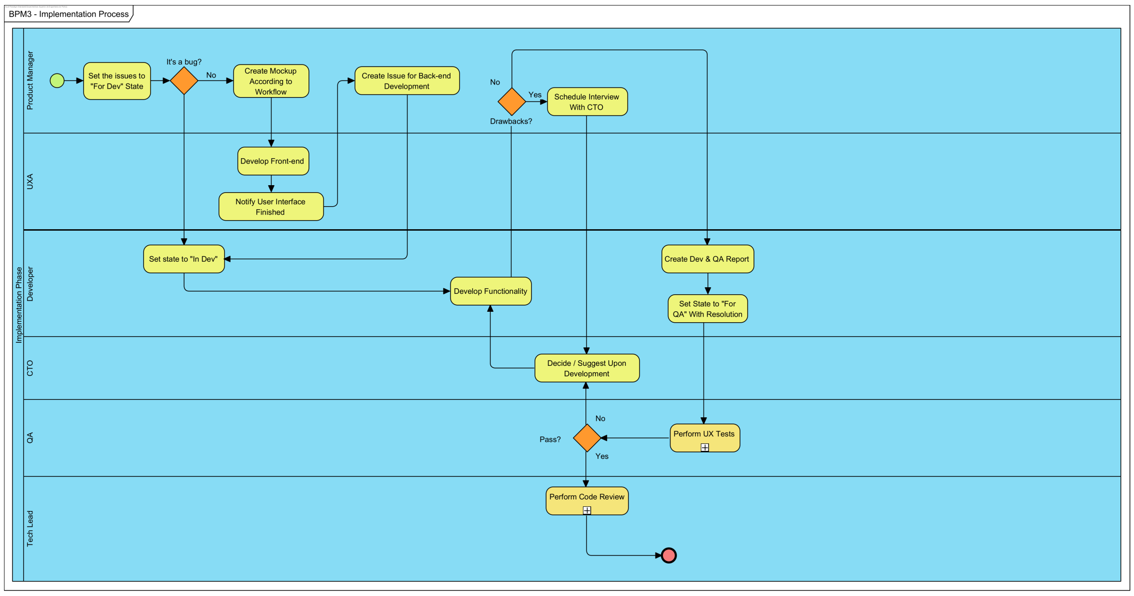

BPM3 - Implementation Process
Product Manager
UXA
Implementation Phase
Developer
CTO
QA
Tech Lead
Set the issues to "For Dev" State
It's a bug?
No
Create Mockup According to Workflow
Develop Front-end
Notify User Interface Finished
Create Issue for Back-end Development
Set state to "In Dev"
Develop Functionality
No
Drawbacks?
Yes
Schedule Interview With CTO
Decide / Suggest Upon Development
Create Dev & QA Report
Set State to "For QA" With Resolution
Perform UX Tests
Pass?
No
Yes
Perform Code Review

Figure 2.7: Development phase process

On the other hand, if the issue is a bug, then it moves to the "In Dev" state in which it is developed by the developer. During this phase, the PM also has the role of monitoring the team in order to answer any questions that arise during development. In case of doubts or major unforeseen events, a meeting is held with the CTO in order to resolve the situation.

After the implementation of the issue, the developer creates the Dev and QA report, where it describes what was done, how it was done, and what and how it should be tested by QA and puts the issue in the state of "For QA", along with the appropriate resolution.

As shown in the Figure A.2 in Appendix A, the issue then passes to QA, where functional tests are performed. If it doesn't pass, the team that developed the request is notified and the issue is moved to the status of "Failed in QA". Otherwise, iteration tests are performed to verify the absence of errors. If there are problems, the issue is sent back to the development team to be fixed. When the functionality is launched for production, the QA is also responsible for analyzing user behavior via Full Story, a tool that allows capturing the customer's interaction with the platform in real time.

If there are no errors, the issue is moved to "For CR" status, where the code review will be done by the TL. As shown in the Figure A.3 in Appendix A, if the code does not pass then the issue moves to the "Failed in CR" state and the developer is notified. Otherwise, the feature is merged in the release branch and the is assigned to the "In Release" state.

2.4.4 Release Phase Process

The release phase comprises three distinct phases: pre-release, technical release, and post-release. In the pre-release (Figure 2.8, first activity), we have the grouping of commits that should and should not go into the release, as well as the definition of epics - bigger and more complex features of the issues developed during the week. The pre-release sub-process flow is shown in Figure A.4 in Appendix A with more detail.

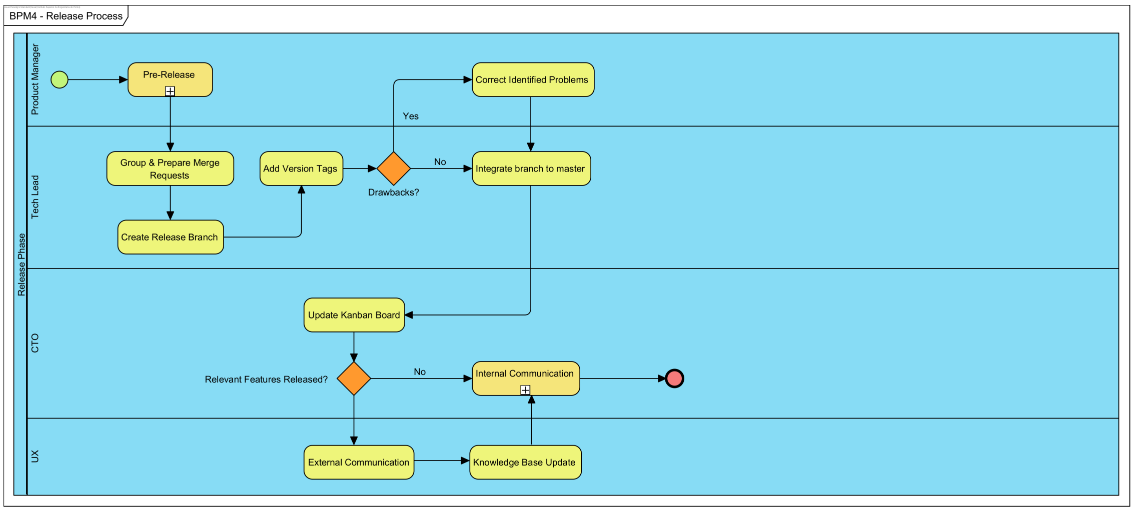


Figure 2.8: Release phase process

In the technical release phase, according to the Figure 2.8, the TL is responsible for grouping the merge requests for a new release branch and adding the respective versioning tags. This phase includes the technical release itself and is carried out every Monday or Wednesday.

Next, there are checks for errors caused by the merge or for other reasons, such as project builds. If there are errors, they are fixed, otherwise, the branch is sent to production. The resolution of errors depends on their severity. If they are difficult to resolve, the deployment to production is postponed and done in a new version. On the other hand, if they are minor, these are corrected on the spot, and the deployment is done.

Finally, after this phase, enters the post-release phase, which comprises internal communication (if they are low-cost functionalities) through the Bitrix24 platform - used by the entire company for internal communication - and later explained in a weekly event held by CTO gauge (as shown in Figure A.5 in Appendix A). On the other hand, if it is a feature of significant relevance, the UXA team is responsible for promoting it on the various E-goi channels, such as blogs, knowledge base, newsletter, etc.

2.4.5 Team Monitoring Process

The team monitoring process is a monthly process that requires a great deal of involvement between the product teams and the respective managers. This is a process that occurs parallel to planning, development and release. It aims to analyze the activities carried out by the teams in order to verify that the established objectives are being met.

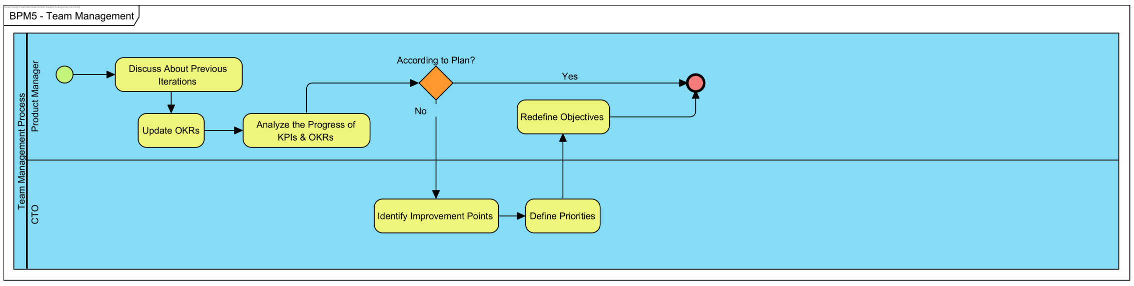


Figure 2.9: Team Management Process

According to the Figure 2.9, this process falls on the masters who are responsible for controlling and monitoring the KPI and OKR of their respective product team. Metrics for each component are calculated and updated in real-time through a script. Otherwise, objectives are manually updated by each manager as they are met.

As each team progresses, the CTO, along with each team's masters, focuses on verifying that the trimester is progressing as planned. The analysis is designed to identify occurrences and unforeseen development deviations so that, if necessary, the planning can be adjusted, focusing on the most urgent matters first.

2.5 Software Development Methodology

In order to more effectively manage the issues necessary to be developed, the company uses a more visual software development methodology - Kanban. This management is done through Jira, which, as previously mentioned, is the tool used by the product department to manage and organize all the tasks.

Regarding the Kanban methodology, it is included in the agile group of methodologies. It provides a means to create a visual representation of the work in progress during the software development process and to limit it. It emphasizes the scheduling of the work in a way that facilitates the delivery of a software product Just In Time (JIT) for implementation [3]. Some of the characteristics that characterize this methodology as per [3] are:

- Maximizes productivity: it delivers an optimized workflow by maximizing productivity and reducing idle time;
- Continuous delivery: it suggests a continuous delivery approach instead of batch releasing in order to release only small parts of software attending the dynamic meetings of the customers;
- Waste minimization: the tasks are only carried out when they are really needed, which results in the minimization of wasted work and time;
- Limits Work In Progress (WIP): its main objective is to limit the WIP in order to optimize the system accordingly with its capacity;
- Kanban board: it represents a workflow visualization tool that guides the workflow and optimizes it by diving it into to-dos, in-progress, and done requirements.

2.5.1 Kanban Board Workflow

The development teams Kanban board is divided into columns according to the possible states of the issues (as per Table 2.2). This board holds two panes, one for the different states of the issues and another for the epic issues (more complex issues).

In the backlog column, are present all the issues that were analyzed and prioritized based on the prioritization system presented in the Subsection 2.4.2. Those with a more pressing priority are moved to the column for development, which contains all the issues that are going to be developed during the week.

By the time the developer responsible for the issue starts resolving it, he drags it to the "In Dev" column, in which case the person that opened the issue is notified by e-mail. By the time that the developer responsible for the issue finishes resolving it, he drags it to the "For

QA" column. By this time, the developer must have also written the development and QA report in order to explain what he did and what/how to test the feature.

After this, the QA team is responsible for analyzing and testing all the functionalities developed by the developers. If any problem arises, it should be dragged into the "Failed in QA" column, in which case the developer must review the comments and act accordingly. The same process happens afterwards, when the QA team puts the issues in the "For CR" column, which is the responsibility of the Tech Lead to do code review and decide upon it.

Lastly, if there are no immediate problems, the issue is moved to "In Release," where all the actions are taken to bring together all the issues that have been resolved and developed to create a revised version of the product.

2.5.2 Issues Management

The issue creation can be made by all the members of the company, that are in direct contact with the E-goi platform in order to aid the product department in bug discovery and to suggest new improvements and features. This report is made in Jira software, however due to the lack of resources in users limitations, the use of the platform could only be used by the development team, this lead to the creation of the "Issue Creator", the project used in order to allow all the collaborators in the company to create issues by piping them to the Jira software, since it is integrated with it. The Issue Creator is a simple User Interfce (UI) that allows users to fill out some fields to open an issue. Even though it prevents multiple limitations, it serves its purpose, despite creating others.

Each issue has the same characteristics and must follow the same workflow. In addition to the priority assigned to an issue, there are two other attributes that are equally important to the company - the status and the resolution. In order to understand the current status of an issue and the conclusions reached after analyzing it, these statements are imperative.

For product management inside the company, transparency of issue states is very important. This provides an overall view, to the Masters, of the current state of each issue under development as well as the amount of work that needs to be done. The Table 2.2, presents each one of these states as well as their correspondent owner (person responsible for the issue in that state) and their correspondent phase in the development.

Table 2.2: Issues status descriptions

Development Phase	**Status**	**Description**	**Owner**
Planning	In Triage	The issue is waiting to be reviewed. Typically, only recently created issues are in this status. This issue is waiting for validation of all information and a decision based on the prioritization model	PM
	Reopened	The issue was once resolved, but the resolution or the work done was not the correct one	PM

Development Phase	**Status**	**Description**	**Owner**
	In Analysis	The issue needs more supporting information before any decision is taken (usually from stakeholders or issue related members)	PM
	On Hold	The issue depends on another development or another team. Dependencies need to be solved in order for the issue to be considered	PM
	Backlog	The issue has been reviewed, prioritized and accepted as a candidate for our roadmap	PM
Development	For Dev	The issue is listed for short-term execution (couple of weeks). The issue is more severe or pervasive than other issues	TL
	In Dev	This issue is being actively worked on at the moment by the assignee	Developer
	For QA	The issue is ready to start the quality assurance tests	QA
	In QA	An action/development for this issue has been proposed and is being reviewed and quality-tested	QA
	Failed in QA	The issue did not passed the acceptance and quality tests and needs additional work	Developer
	In CR	The issue is being reviewed by the tech lead to ensure the patterns	TL
	Failed in CR	The issue did not passed the code review and needs additional work	Developer
Release	In Release	A resolution has been taken and is waiting to be shipped in a release. It's not guaranteed that it will be shipped	PM
	Release	The issue is considered finished, the resolution is correct and shipped in a release	PM

On the other hand, we have issue resolution that corresponds to the decision made regarding the issue after it has been analyzed and/or developed. The issue resolution process is important from a business standpoint as it allows the collaborators to verify that the problem has been analyzed and solved. The Table 2.3, lists all the possible resolutions that an issue can take part in.

Table 2.3: Issues resolutions descriptions

Resolution	**Description**
Unresolved	Default resolution
Done	The issue is solved, checked into the release and tested
Done - No backwards compatibility	The issue is solved from the resolution date forward. It will not solve the previous cases
Done - In another issue	The issue was solved by doing another related issue
Done - With Reserves	The issue is partially solved or solved but not tested properly. A fail-safe may have been developed to prevent this
Won't Do	The issue will not be solved anytime soon
Won't Do - Cannot Reproduce	The issue was not possible to reproduce in any environment (development, Staging and Production). All attempts at reproducing this issue failed, or not enough information was available to reproduce the issue. Reading the code produces no clues as to why this behavior would occur. If more information appears later, please reopen the issue
Won't Do - Not a Bug	The issue has the supposed behavior. This could lead to an improvement or further discussions
Won't Do - Deprecated	The issue is related with a feature, or layer will no longer be supported for new improvements or features. This could lead to a different approach
Won't Do - Unjustifiable	The issue has a huge effort in relation with the outcome, or it's not strategically viable
Won't Do - Lack of information	The issue has not enough information to be pursued. All attempts were made to pursue information with the reporter and stakeholders
Won't Do - Incident Related	The issue was created due to an incident or a temporary glitch
Duplicated	The issue is a duplicate of an existing issue

2.6 Summary

In this chapter, were presented an overview of the company structure as well as the processes inherent to the product department in the development of updated product versions. In the same vein, the roles of the members of each team and the hierarchy of such and the methodologies used to gather costumer feedback inside E-Goi was explored.

By conducting a process overview, it is possible to identify the opportunities and threats available in these processes. This will enable one to develop an understanding of the requirements needed to develop a solution that can overcome these problems.

The next chapter includes the solutions explored by other authors in the fields of issue management as well as other relevant techniques related to the development of the project.

Chapter 3

Literature Review

To be able to follow the most effective approach to solving the problem at hand, a prior analysis of the theoretical background that will allow the retrieval of the necessary information is made. The scientific areas to be researched are related to issue-reporting standardization, issue duplication detection techniques, customer involvement in the development process, issue tracking systems (with a comparative analysis between them), and BPMN.

3.1 Issue Standardization

To comprehend what elements are relevant for the uniformity of the submission of new issues, we must first comprehend what is an issue. As stated in [4] an issue represents a bug, a task, or other issue types (feature, improvement, task, etc) in a particular project. In most cases, issues present a few default, mandatory fields that allow their management, including issue type, priority, labels, complexity, summary, description, resolution, and status.

For software development, bug and issue reports are crucial since they allow users to inform developers of problems, improvements, and new features. Issue reports typically contain the details of the actions needed in order to point the developers in the right direction, in the eyes of the reporter. These should define a reasonable amount of information, so that is not too demanding for the reporters but enough to provide the needed information to developers [5]

Bug Reports

In the case of the bug report, however, these reports vary drastically in terms of content quality in the sense that they often provide incorrect or insufficient information [6]. So, developers often face themselves with descriptions such as "Alterar para lowercase bounce returnpath" (E-goi Jira, Bug #BB-17756) or "Pago Plano + Addon, mas addon não atribuído" (E-goi Jira, Bug #BB-18015). This problem slows down the developer's work and the resolution of the issue. This is because the developer has to request additional information from the issue reporter to be able to resolve the issue.

Bug reports often contain multiple information, such as reproducing steps, test cases, crash stack traces and fix suggestions, summary, and others. Thus, to make reports consistent, templates are provided where certain required and optional fields are specified to be filled by the reporters [7]. When very little data is provided, it is extremely difficult for developers to reproduce the errors reported and, therefore, proceed to resolve them.

So, to decide which fields are the most pertinent to include, we must consider which ones are more valued by the developers that treat those bugs. According to [7] when selecting the bug report to process, the developer's first action is to try to reproduce the problem at hand. This step is critical not only to verify that the problem exists but also to be able to more easily evaluate the fix that they provide. Also, other information such as stack traces, crash descriptions, observed behavior, and expected behavior are equally important to achieve the proper issue correction [6].

According to [8–10] the sections that the developers find most useful when fixing bugs, with their rank in importance, are the following ones:

1. **Steps to reproduce**: comprises a clear set of instructions that the developer can use to reproduce the bug on their own. The set of operations must be clear enough so that the developer can reproduce the error on his machine;
2. **Stack traces**: a stack trace produced by the application, most often when the bug is related to a crash;
3. **Test cases**: steps that a developer can use to determine when they have fixed the bug;
4. **Observed behavior**: the behavior of the application that the user saw happening as a result of the bug. The statement should be informational and clear enough and not contain simple statements such as "isn't working" or "nothing happens";
5. **Screenshots**: screenshots of the application while the bug is happening;
6. **Expected behavior**: what the user expected to happen in the application on the contrary of the observed behavior;
7. **Code examples**: an example of some code that can cause the referenced bug;
8. **Summary**: a short (usually one-sentence) summary of the bug. This field is mandatory when writing a bug report. In fact, this field is often used to detect similar and duplicate bugs [9];
9. **Version**: what version of the application the user was using at the time of the error (if applicable);
10. **Error reports**: an error report produced by the application as the bug occurred.

Although the features reflected above are the main ones that seem to be valued by the developers, other information can be included in order to also help the people from QA and managers such as component, business information, additional information, and workaround [10]. The fields needed to submit new bug reports vary according to the needs of the projects and the teams.

Feature and Improvement Requests

Feature requests, also referred to as JIT requirements or enhancements, comprise a structured request (an issue report with a summary, description, and other attributes), that documents an adaptive maintenance task whose implementation results in enhanced functionalities, normally per user feedback [11].

A feature request usually relates to a single requirement. A user can specify them, or even a developer can, after which they are analyzed and verified if and when they will be implemented. The implementation of the requests depends on the priorities set by the users (number of requests), developers, or managers [11].

Accordingly to [11], to submit a high-quality feature request one must follow the following quality criteria:

- **Completeness**: all needed elements must be present, regardless of whether they are basic, required, or optional. For the basic elements, we have the summary, description, product version (if applicable), and relative importance. For the required ones we have: keywords, rationale, and link to code (if applicable), and for the optional we have: use case scenarios, screenshots, and possible solutions;
- **Uniformity**: the style and format should be standardized. This leads to reduced time spent understanding and managing requirements. This can be achieved with the use of appropriate tools;
- **Conformance**: the requirements should be consistent and correct. This includes the use of correct language, specification of the problem identified, an accurate summary, no duplicates.

These quality attributes allow the definition of the elements that must be present in a feature request report. The Table 3.1 specifies the elements, their description, and their priority to be included in a final report.

Table 3.1: Feature request report elements

Element	**Description**	**Priority**
Summary	A brief description of the feature request that uniquely identifies the desired behavior	Required
Desired behavior	The detailed workflow of the desired request. The steps the functionality should follow	Required
Goal	Detailed description of the importance of the desired request and how it will improve the application	Required
Additional information	Comprises any additional information that may help in the implementation of the functionality, such as references or links to other implemented functionalities of the same sort	Optional
Screenshots and other attachments	Complements the previous element	Optional

As it occurs with bug reports, these elements must also be adapted for the teams and work methods of those teams. So other information can also be included, such as business information, components attribution, etc.

On the other hand, the improvement request follows most of the guidelines of the feature requests. According to [4], an improvement is an improvement or enhancement to an existing feature or task. In contrast with the feature request, an improvement request implies that additional elements beyond the ones referred to for the feature request (in

Table 3.1) must also be included, namely the current behavior, which describes the current, observable behavior that needs improvement.

3.2 Issue Duplication

Another problem regarding issue management, beyond uniformity, is the problem of issue duplication. This problem is characterized by the potential submission of more than one report describing the same change request [12]. The main consequence of this problem is the additional work required by the triaging team elements to identify these duplicates.

Triaging is a process in which the person delegates the issue to a particular person for its resolution [13]. Before assigning, the triager needs to read through the reports in order to find which ones are duplicates and which ones are not. Sometimes the same problem is assigned to more than one person, which leads to the waste of resources. This leads to redundancy in the work and thus increases the workload for engineers [13].

Issue Duplication Causes

Even though many factors can contribute to the creation of duplicate issue reports, according to studies performed by [6, 14, 15], some of these factors can be:

- **Laziness and lack of experience from users**: some users are not well known with issue trackers, while others simply are not willing to spend time searching for the reported issue;
- **Poor search features**: lack of functionality in the search feature of the used tools;
- **Multiple issues, one reason**: sometimes the same reported issue can be related to the same cause, which is not always clear for the reporters;
- **Intentional re-submission**: some users intentionally submit the same report in order to try to reach its resolution;
- **Accidental re-submission**: most of the issues submitted are due to errors in the platform or the fact that two submitters report the same issue at the same time;
- **Lack of uniformity**: the lack of common vocabulary and consistency in the submission of the reports causes some of the duplication tools to not be able to process correctly such reports.

Issue Duplication Detection Methods

Most of the literature regarding the duplication of issue reports can be divided into two categories. One is the prevention of duplicate issue reports while submitting - duplicate prevention - and the other is the detection of duplicates during the triaging phase - deduplication [13].

Deduplication methodologies also known as duplicate report detection, have been studied and implemented by several authors [16]. They consist of querying almost identical reports to group them. Most of these methodologies are based on machine-learning, topic analysis, deep learning and information retrieval [16]. Information retrieval refers to the activity of obtaining the required information from a group of document resources. They are applied to various scopes such as image retrieval and web searches [17].

Regarding the methodologies used to detect duplicates before the report submission, information retrieval is the most commonly used. In fact, the author [16], suggests a methodology known as "Continuous Querying".

This technique consists of providing multiple suggestions to the user as he writes his report, by using machine learning algorithms to retrieve the top-n most similar issues to present these suggestions to the user. As the user types in the information, either in the description or the summary, queries are made to the issue tracker to find any identical reports. The information is then presented to the user, who can pause the writing of the issue and inspect any of the suggested duplicates. Using this method, the authors found that 42/

3.3 Issue Tracking Systems

Issue Tracking System (ITS) are a common standard in most software development projects. They allow both developers, testers, managers, and users to submit issue reports or other tasks [18–20].

These systems are not just a database to store and track bugs, features, and tasks but also the main means of internal and external communication and work coordination for many software development teams [20]. Because these systems are mainly used for project management, there is, in most cases, high integration with other development and management tools [18].

On the topics bellow, is presented a brief description of each of the three most used and referenced ITS and presented a feature-based matrix of comparison between the several relevant features of them. This includes Jira, which is the ITS used by E-goi development teams.

Jira

Jira [21] is a bug/issue tracking and project management system developed by Atlassian Software Systems as an open-source code project. It provides complex features at the same time, with an easy-to-use interface.

Jira provides numerous plug-ins and extensions developed by its community and an extensive variety of documentation and learning material. Atlassian provides Jira for free to open-source projects and small teams (10 users), however, it is paid for commercial uses.

Redmine

Redmine [22] is free and open-source, written in Ruby, and provides both the functionalities of issue tracking and project management. It integrates with various version control systems and is cross-platform and cross-database. As it happens with Jira, it also provides support to add new plugins as required.

Bugzilla

Bugzilla [23] It is used on many open-source projects such as Mozilla, Eclipse, and many Linux distributions, and is well suited to large, collaborative projects. It is a free and open-source tool that possesses almost all the functionalities of the previous tools except for project management and official plug-ins with cross-platform integration.

According to information retrieved by the articles [21–25] a feature-based matrix is presented in Table 3.2 to compare the different features available on the three platforms.

Table 3.2: ITS feature-comparison matrix

Feature	**Bugzilla**	**Redmine**	**Jira**
JIT Duplicate Retrieval	Yes	Yes, through plugins	Yes, as additional plugin
Bitrix Integration	No	No	No
LiveAgent Integration	No	No	Yes, through plug-ins
Advanced Search	Yes	Yes	Yes
Field Customization	Yes	Yes	Yes
Notifications	Yes	Yes	Yes
Version Control Integration	Yes	Yes	Yes
REST API	Yes	Yes	Yes
Edit Conflict Warning	Yes	Yes	No

The analysis of the presented table shows that all three IST provide almost the same features. However, Jira allows the connection with the LiveAgent platform, even if that is done through plugins. Although the use of any of the presented IST could in fact serve as a valid alternative to the actual platform used in E-Goi (Jira), one thing that must be taken into account is that none of them provides any form of integration with the Bitrix24 platform, which could pose an obstacle to the requirements needed by the stakeholder of integrating the development process life-cycle into this platform.

3.4 User Involvement in Software Development

In the last years, a new trend in requirements engineering is to motivate software evolution by gathering and analyzing user feedback from issue tracking systems [26]. The ultimate success of any software depends on whether or not the developed solution manages to fulfill the expectations of the final users [27].

Regarding the development of new services, two main operational outcomes must be considered - operational efficiency and market competitiveness. In one sense, operational efficiency relates to the speed of innovation and technical quality and is calculated by measuring the effort on an internal basis. On the other hand, market competitiveness relates to market superiority and sales performance and is measured by the success achieved, which is calculated in an external perspective [28].

In Agile methodologies, since they provide iterative development, they allow an intensive involvement of the end-user [27]. This methodology allows the involvement of the end-user in the development of new features more efficiently, rapid feedback gathering cycles, and functionality prioritization. User involvement grants several advantages such as improved quality due to more precise requirements and the reduction of waste by preventing the development of non-required expensive features. This, as a consequence, enables the companies to develop their products with bigger success rates [28].

On the other hand, the lack of customer collaboration in agile teams [29], revealed to lead to several consequences such as pressure to over-commit, difficulty in rounding and deciding upon requirements, requirements prioritization issues, loss of productivity, and, in rare cases, business loss. For this reason, customer involvement is essential to improving performance in the development of new services [28].

3.5 Business Process Model Notation

Originally published in 2004, BPMN defines a diagram based on a technique of workflow development, through graphical schemes that translate to operations of business processes. The primary goal of this notation is to provide a graphical language that is quickly understandable by business users, from the business analysts who are tasked with drafting the processes to the developers responsible for implementing them [30].

This notation offers multiple benefits to the companies where it is implemented, such as bigger visibility across the company of its activities and processes, process execution time reduction, better chances to detect and correct poorly optimized processes and a better understanding of the tasks and responsibilities of each collaborator inside the company [31].

By being a graphical notation, BPMN comprises four categories of graphical elements that help build the diagrams [30]:

- Flow Objects: represent all the actions that can happen in a business process, which determine their behavior. They include events, activities, and gateways;
- Connecting Objects: represent the message exchange between the flow objects. They include three types of connections - sequence flow, message flow, and association;
- Swimlanes: they give the ability to group several elements into groups of actions performed by the same intervenient;
- Artifacts: are used to provide additional information about the process.

3.6 Summary

In this chapter, all the information obtained and necessary for the resolution of the detected problems was referenced. Several topics were discussed, such as the standardization of issue reports and the reasons why most reports do not contain consistent information. The most common causes for the appearance of duplicate issues in its analysis and the most effective methods for their detection and reduction were also referenced. Finally, an evaluation of different issue management platforms was also made, as well as the importance of the feedback acquired from the final consumer for the improvement of the product.

In the following chapter, an analysis of the value of the project will be carried out. In addition, it will be presented as a presentation of the requirements elicited in the interviews carried out with the stakeholder.

Chapter 4

Analysis

This chapter analyzes the problem at hand. Not only does this analysis consider the value of the proposed solution, but it also examines the value it will bring to the company, and consequently to the customer and the requirements engineering section as well. In this later section, the requirements gathered through elicitation are presented as well as the non-functional requirements that translate into quality attributes.

4.1 Value Analysis

In accordance with [32], value analysis correlates to the use of specific techniques, knowledge, and skills that allows the efficient discovery of unnecessary costs on a product, i.e. that lacks customer features or quality. In general, it is defined as a method applied in order to systematically improve the value of products, services, and processes [33].

Thus, to idealize, design, and implement a new product, a value analysis process must be conducted. This analysis should, not only, cover the company's need for the product, its costs, and benefits, but also the value that it will bring to the customer. In this section, the value analysis of the product to be developed will be addressed in more detail.

4.1.1 Innovation Process

Innovation can be described as "change and process improvement for a product or system in a way that this change will be new for the firm" [34]. Innovation can bring several benefits such as improved productivity, reducing costs, increasing competitiveness, among others and a high innovation rate usually translates to higher profitability rates in the long-term [35].

Accordingly to [36], the process of innovation can be classified into three main areas:

- Fuzzy Front End (FEE)
- New Product Development (NPD)
- Commercialization

Regarding the FEE, it represents the initial phase of the innovation process. Its objective is to generate an idea that will either be approved for development or cease. Thus, to achieve that goal, it comprises two main sequential actions to be performed - idea generation and assessment, which allows the definition, exploration, and selection of the new ideas; and concept development and product planning, which results in the planning of the next sequential phase which is the NPD [37].

On the other hand, the NPD phase entails the transformation of the idealized concept into an actual product, which includes different actions such as design reviews, market tests, prototype development, and product testing and redesign [37].

Finally, the commercialization phase corresponds to the production of the projected product, market introduction, and penetration, and finally, the monitoring of the results and continuous product verification.

4.1.2 New Concept Development

Despite the fact that the FEE represents a valuable opportunity for innovation process improvement, the fact that the front end does not support a common language of the key elements of the front end makes it difficult to compare and find the best practices to guide the FEE [38]. This, in countermeasure, leads to the research and development of the New Concept Development (NCD).

NCD consists of a model that aims to resolve the gap that exists in the need for a standard language, concepts, and best practices for the FEE phase. The NCD comprises three essential parts [36]:

- **Engine**: or central point corresponds to the portion that drives the five key elements and that is conducted by the leadership and organization culture;
- **Activity Elements**: consists of five of them - opportunity identification, opportunity analysis, idea generation, and enrichment, idea selection, and concept definition;
- **Influencing Factors**: the environment on the outside of the organization. Consists of organizational capabilities, business strategy, the outside world, and the science that will be utilized [38].

In order to apply the NCD model to the proposed solution developed in the organization, each of the five key elements will be discussed below.

Opportunity Identification

Identifying the opportunity that the organization may want to pursue, usually aligned with its business goals, constitutes this step. Opportunities can arise in a variety of forms such as a response to the competition, a way to achieve a competitive advantage, to simplify operations, or even to reduce the production costs [36].

As stated previously, in Section 1.2, several issues were identified that needed addressing in such a way that it improves the efficiency of the processes in product development and costs derived from used tools.

Opportunity Analysis

This element ensures that the opportunity addressed in the previous point is worth being further explored, even though that additional analysis is required in order to successfully translate the opportunity into business value.

Since the main objective of this element is to gather the necessary information regarding the market and competitors a Strengths, Weaknesses, Opportunities and Threats (SWOT) analysis can be performed. This analysis includes identifying the internal strengths and weaknesses as well as external opportunities and threats. These are factors that can affect the devised product. This analysis is present in the Figure 4.1.

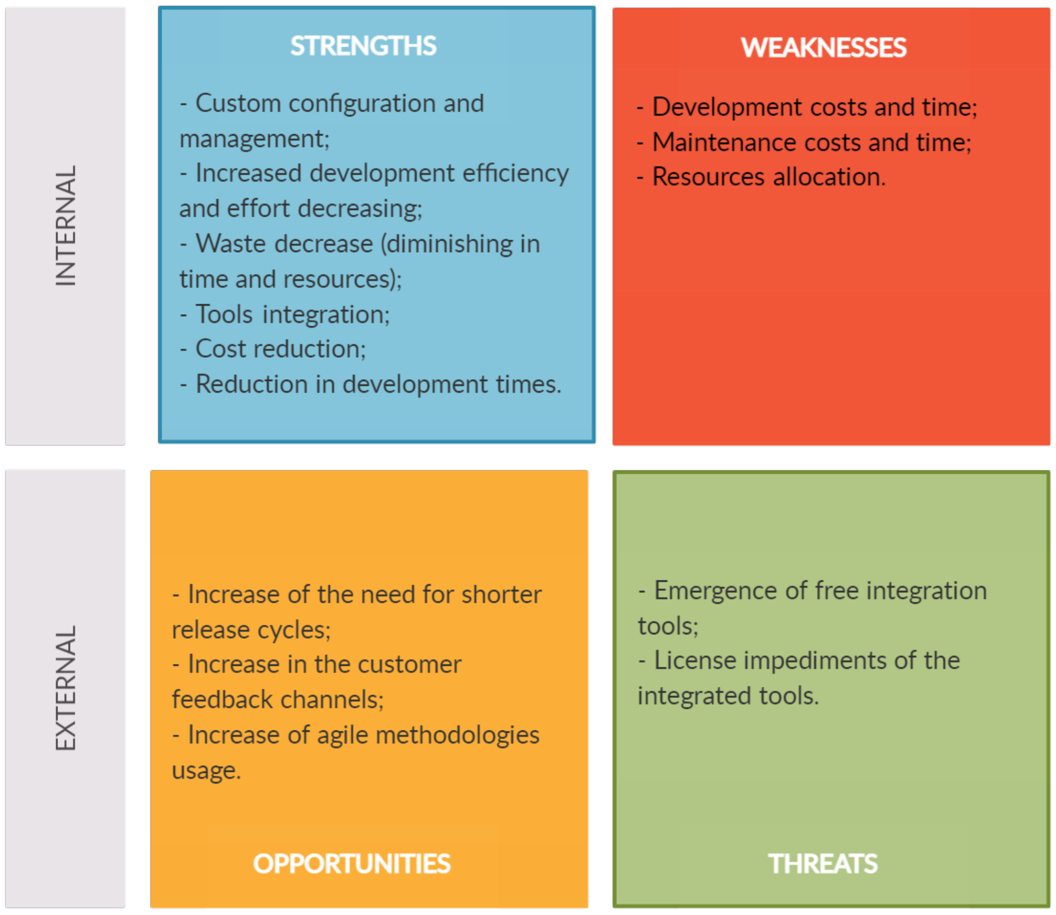

INTERNAL
STRENGTHS
- Custom configuration and management;
- Increased development efficiency and effort decreasing;
- Waste decrease (diminishing in time and resources);
- Tools integration;
- Cost reduction;
- Reduction in development times.
WEAKNESSES
- Development costs and time;
- Maintenance costs and time;
- Resources allocation.
EXTERNAL
- Increase of the need for shorter release cycles;
- Increase in the customer feedback channels;
- Increase of agile methodologies usage.
OPPORTUNITIES
- Emergence of free integration tools;
- License impediments of the integrated tools.
THREATS

Figure 4.1: Product SWOT analysis

As a first step, we have strengths that provide the characteristics that will add value to the organization, specifically to the development teams. These characteristics include improving the efficiency of the development team and, consequently, reducing costs and waste. On the other hand, we have weaknesses that will negatively impact the solution inside the company such as the development costs and time required to build the platform and regular maintenance associated.

As for the opportunities, we have the increasing of the customer feedback channels may it be direct or indirect which, in turn, increases the amount of feedback (problems and features requests), which directly impact the amount of work to be developed and the decreasing of the development and releasing cycles to bring value to the customer more rapidly and be able to encompass the constant market threats. In contrast, in threats, we often see the emergence of new free tools that can render the product useless, and new license restrictions on integrations that can complicate the use of certain features.

Shortly, it is possible to demonstrate that the development of the solution can, in fact, deliver some benefits that will, directly and indirectly, enhance the customer experience. These benefits can indeed justify development costs and resources.

Idea Generation and Enrichment

This phase consists of the creation, development, and transformation of the opportunity into a concrete idea. In an effort to resolve the identified issues and meet all the requested requirements. After some brainstorming, and taking notice of the cost and development constraints, three options were identified:

- Development of a solution from scratch that can both enforce uniformity in the issue-reporting process and perform integration with the other tools. This reduces the costs related to the use of Jira for all the users but also creates the costs of development and maintenance;
- Creation of the users in Jira, the definition of the reporting model templates and enforcing them, and using other software to perform the integration with the applications. This involves costs associated with the use of another tool for integration (both in configuration and for using the tool);
- Taking care of the uniformization problem using Jira but disregarding the tools integrations.

Even though all the ideas fulfill the requirements needed, except for the last one which does not take into consideration the integration with the tools, it is necessary to perform an analysis on the viability of each of them to select the one that best suits the company needs while creating value to the customer. Thus, this selection is made in the following phase of NCD.

Idea Selection

This phase of the NCD consists of the analysis of the different ideas and the selection of the ones to pursue to achieve the most business value (more on this subject in Section 4.1.3) [36]. The selection may be as simple as an individual choice or formalized within a complex business process.

One methodology that can be employed to achieve this selection is the Analytic Hierarchy Process (AHP), which consists of a theory of measurement that allows a user to make pairwise comparisons based on numerical decisions from a pre-specified scale of numbers [39]. The model is one of the most widely used decision-making tools, used in the widest range of fields like planning, selecting the most appropriate alternative, resource allocation, conflict resolution, optimization, etc [40].

The following paragraphs will be devoted to applying the AHP method to the problem at hand to find the most suitable alternative idea from those suggested during phase one. Its important to reference that all the criteria and relative importance values given throughout this analysis was previously analyzed and revised with the CTO, which means that the conclusions taken take into account the necessities of the stakeholders.

The first step in AHP consists of the definition of the decision-making elements which consist of the problem statement, the alternatives which were already devised earlier, and the criteria associated with each one of the alternatives. These are presented in the hierarchical tree, shown in Figure 4.2. The alternatives consist of the following (the letters associated with each alternative and criteria reflect the ones used from this point forward in each of the presented tables):

- **X**: Resolve the uniformization problem using Jira but disregard the tools integrations;
- **Y**: Develop a customized solution from scratch;
- **Z**: Create the users in Jira and enforce uniformization through templates and integrate using existing tools;

As for the criteria that will allow an improved understanding of what description better relates to each of the alternatives, these are:

- **A**: Lack of integration
- **B**: Customization
- **C**: Cost

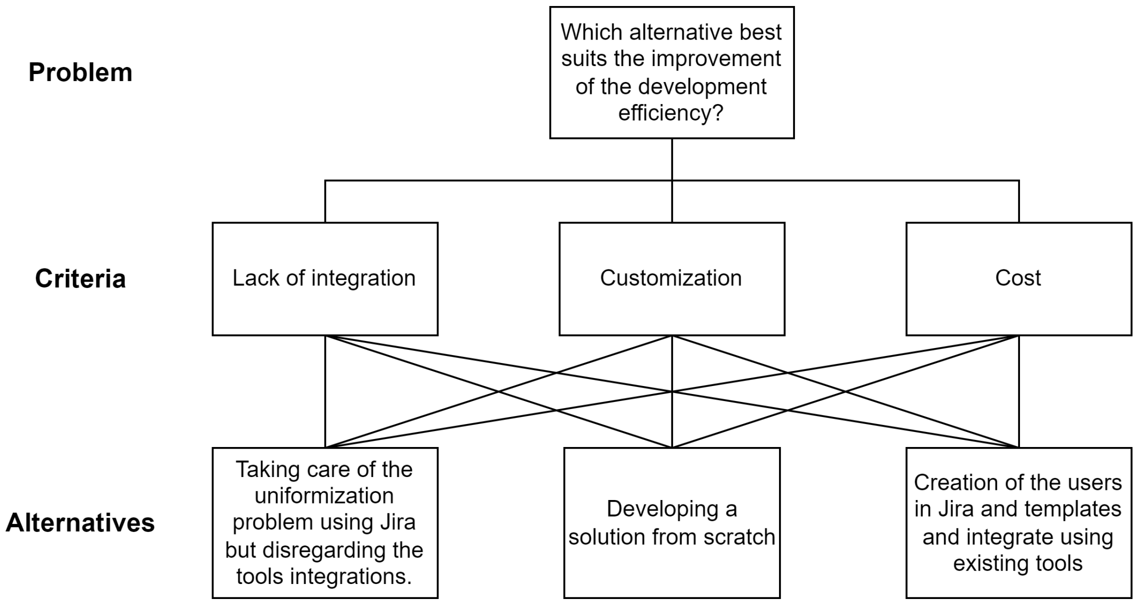

Problem
Which alternative best suits the improvement of the development efficiency?
Criteria
Lack of integration
Customization
Cost
Alternatives
Taking care of the uniformization problem using Jira but disregarding the tools integrations.
Developing a solution from scratch
Creation of the users in Jira and templates and integrate using existing tools

Figure 4.2: Comparisson tree of the problem

In the second phase, alternatives and criteria are compared by means of a comparison matrix created after the problem statement is defined. The priority assigned to each one of the attributes is defined in a Fundamental Scale. In this scale, the number 1 gives equal importance to the attributes, and 9 represents the maximum importance. Intermediate values can be proposed with a range of importance ranging from 1 to 9.

The priority matrix is presented in Table 4.1, compares all the alternatives in terms of their importance and the sum is presented which is later used to calculate the normalized matrix. For each interception of attributes (A with B, A with C and B with C) we present a value from 1 to 9 according to the relative importance that one has over the other, on the remaining slots we calculate the inverse of that value, for example, in the interception of A with B which is of value 7, the inverse will be 1/7 which is 0.14.

By analyzing the table, we can verify that the factor "customization" is more significant than both the lack of integration and the cost of implementation, although the cost is more important than the lack of integration.

Table 4.1: Comparison matrix for the criteria and respective column sums

-	**A**	**B**	**C**
A	1,00	0,14	0,20
B	7,00	1,00	3,00
C	5,00	0,33	1,00
Sum	13,00	1,48	4,20

Next, to calculate the relative priority of each criterion we must create the normalized matrix to match all the criteria to the same unit. To do so, each value of the matrix (Table 4.1) is divided by the sum of its respective column. Table 4.2 shows the normalized matrix of the original one.

Table 4.2: Normalized matrix with relative priorities

-	**A**	**B**	**C**	**Relative Priorities**
A	0,08	0,10	0,05	0,07
B	0,54	0,68	0,71	0,64
C	0,38	0,23	0,24	0,28

By analyzing the Table 4.2, we verify also the presence of the relative priorities vector, which was calculated by doing the arithmetic average of the values of each one of the normalized matrix rows. This vector corresponds to the order of importance of each one of the criteria.

With the analysis of the values, we can verify that the criteria "customization" is the one that is considered to be the most relevant for the problem at hand. It makes sense since a solution that is low cost as well as allowing customization of both its features and tools integration is more likely to add value to the organization.

Even though all the necessary steps to calculate the priority vector were correctly performed, the use of these values is still dependent on a consistency validation. This is to measure how consistent the judgments are. To do so we must calculate the Consistency Index (CI) to be able to determine the Consistency Ratio (CR). The CI is given by the formula:

$$CI = \frac{(\lambda_{max} - n)}{(n-1)}$$

To get the λ_{max}, first, we multiply each one one of the values in the original matrix by the correspondent relative priority and sum the values of each row to get the weighted sum (Sum_i) and then divide by the correspondent weighted criteria (W_i), as per Table 4.3.

Table 4.3: Matrix maximum own value calculation

-	**A**	**B**	**C**	Sum_i	W_i	Sum_i / W_i
A	0,07	0,09	0,06	0,22	0,07	3,01
B	0,52	0,64	0,85	2,01	0,64	3,12
C	0,37	0,21	0,28	0,87	0,28	3,06

Thus, λ_{max} will be given by:

$$\lambda_{max} = \frac{(3,01 + 3,12 + 3,06)}{3} = 3,07$$

And the consistency index by:

$$CI = \frac{(3,07 - 3)}{(3 - 1)} = 0,03$$

With this, we can now calculate the CR with the formula:

$$CR = \frac{CI}{RI_{n=3}} = \frac{0,03}{0,58} = 0,06$$

,where RI represents the randomness index that can be retrieved in the Table 4.4, for n=3.

Table 4.4: The values of Random Consistency Index

Dimension	**RI**
1	0
2	0
3	0,5799
4	0,8921
5	1,1159
6	1,2358
7	1,3322
8	1,3952
9	1,4537
10	1,4882

If the CR were to be superior to 0,1 then our judgments would not be trustworthy, since they were too close to the comfort of aleatory, which would mean that our results didn't present consistent values.

Since 0,06 < 0,1 we can confirm that our values are consistent and thus proceed to the next step, which is to calculate the comparison matrix for each one of the criteria, considering each one of the selected alternatives. These matrices and their correspondent relative priority vector is presented in the Tables 4.5, 4.6 and 4.7.

Table 4.5: Comparison matrix for the criteria "Lack of Integration"

-	**A**	**B**	**C**	**Relative Priorities**
A	1,00	9,00	9,00	0,82
B	0,11	1,00	1,00	0,09
C	0,11	1,00	1,00	0,09

Table 4.6: Comparison matrix for the criteria "Customization"

-	**A**	**B**	**C**	**Relative Priorities**
A	1,00	0,17	0,33	0,09
B	6,00	1,00	5,00	0,71
C	3,00	0,20	1,00	0,20

Table 4.7: Comparison matrix for the criteria "Cost"

-	**A**	**B**	**C**	**Relative Priorities**
A	1,00	5,00	0,33	0,28
B	0,20	1,00	0,14	0,07
C	3,00	7,00	1,00	0,64

The final step consists of the calculation of the composed priorities by multiplying the priority matrix with all the values from the relative priorities of the solutions (Tables 4.5, 4.6 and 4.7) and the relative priority vector of the criteria calculated initially (Table 4.2):

$$\begin{bmatrix} 0,82 & 0,09 & 0,28 \\ 0,09 & 0,71 & 0,07 \\ 0,09 & 0,20 & 0,64 \end{bmatrix} x \begin{bmatrix} 0,07 \\ 0,64 \\ 0,28 \end{bmatrix} = \begin{bmatrix} 0,20 \\ 0,48 \\ 0,32 \end{bmatrix}$$

With this calculus, the final global weight of each solution relative to the criteria would be:

- **X**: 0,20
- **Y**: 0,48
- **Z**: 0,32

Trough the evaluation of the final result, since the alternative **Y - Develop a customized solution from scratch** is the one with the highest value, we can verify that this is the best alternative for the solution.

Concept Definition

Finally, the concept definition consists of the proposition and definition of the "win statement", which the company will support to make the final decision whether or not to invest in the proposed solution.

Thus, the concept that the chosen solutions grasp is the development of a fully integrated solution that can be used to optimize product development teams' efficiency and reduce waste, impacting the problems identification phase and improving software release cycles.

4.1.3 Solution Value

There is not much consensus on what value creation is and how it can be achieved. Accordingly to [41], one can differentiate two types of value. The first, use value, which consists of the value of a new service or product that is perceived by users relative to their needs. These judgments are subjective and change from individual to individual.

The other type is exchange value. This represents either the monetary value made at a certain point in time or the amount received from the user when he used the product or service.

Thus, when put all together, we can see that the process of value creation depends on the perception that the user makes of it, who is, ultimately, the judge of the product value. The question then arises as to when the monetary exchange rate will be enough to capture the value perceived, raising two important economic conditions that a company must endure in order to successfully benefit from its value creation processes.

First, the price from the exchange must exceed the production costs and the other is that the amount of money that a user will pay for a product or service is a "function of the perceived performance difference between the new value that is created (...) and the target user's closest alternative" [41].

This subject is relevant for the problem in the way that, in E-goi, the activities that are performed inside the company have a common object- the creation, direct or indirectly, of value to the customer. In the case of product development, this is also true.

All the improvements and corrections are ultimately made to improve the perceived value that the end-users have of the product and, thus, allowing the generation of exchange value that is used to cover the production costs.

Thus, by improving the performance of the development cycle, we are generating more value for the product by reducing the time needed in order to release a new version to the customers.

Value Chain

A value chain consists of a set of activities that a company possesses and uses to create value for its customers. In 1985, Michael Porter proposed a value chain that organizations can use to examine their activities and see how they are connected [42]. This value chain can be useful to examine how an organization can increase its value or reduce costs by aggregating its activities/processes into well structured activity groups.

The premise here is that the more value an organization creates, the more profitable it will consequently be. So when a company provides more value to its customers than, in theory, the more competitively advantageous it will become.

The model consists of five primary activities and four support activities. Primary activities are mainly related to the creation or delivery of a product or service. These activities include the following:

- **Inbound Logistics**: related to the processes that manage activities like receiving, storing and distributing inputs;
- **Operations**: conversion of raw materials into finished goods or services through manufacturing (or transformation);
- **Outbound Logistics**: delivering of goods or services to the customer;
- **Marketing & Sales**: processes used to identify opportunities and processing customer orders;
- **Service**: activities related to maintaining the value of the product after it has been purchased. Providing after-sales support to customers.

These primary activities are facilitated by a group of support activities that span across the entire organizations, which are:

- **Firm Infrastructure**: organization-wide administrative and management systems;
- **Human Resource Management**: human-resources management;
- **Technology Development**: research and development and continuous enhancements of technology-related activities;
- **Procurement**: management of purchases of materials and equipment.

The margin is related to the difference between the value that is created and captured by the organization and the cost of creating such value. In the Figure 4.3 is represented the value chain for E-goi and its activities.

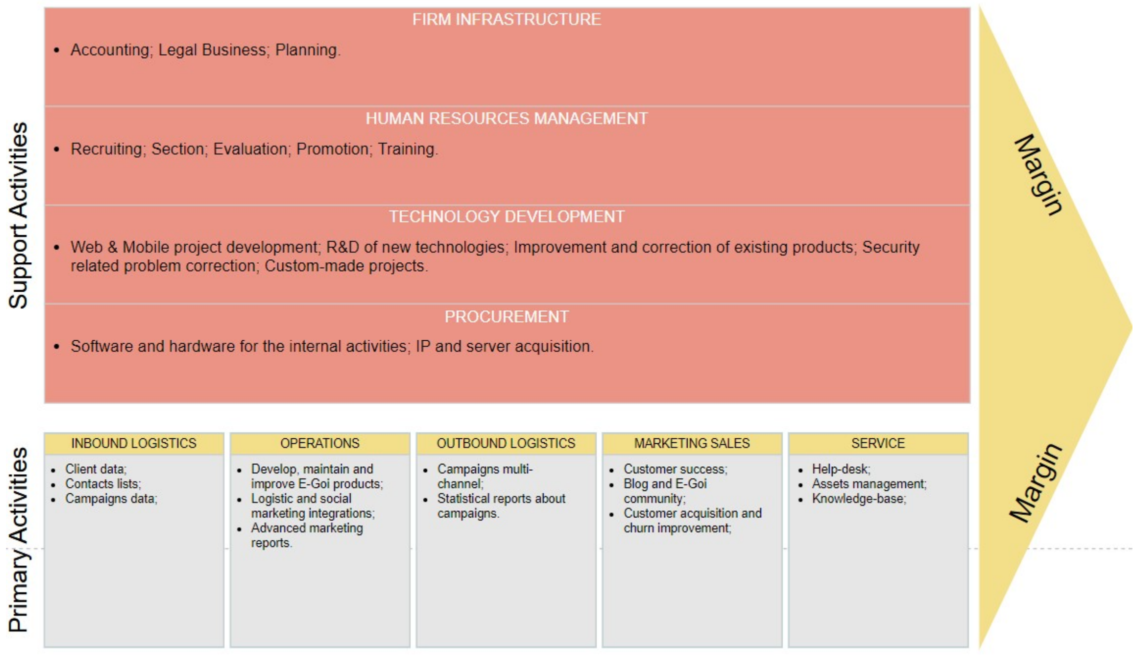

Support Activities
FIRM INFRASTRUCTURE
• Accounting; Legal Business; Planning.
HUMAN RESOURCES MANAGEMENT
• Recruiting; Section; Evaluation; Promotion; Training.
TECHNOLOGY DEVELOPMENT
• Web & Mobile project development; R&D of new technologies; Improvement and correction of existing products; Security related problem correction; Custom-made projects.
PROCUREMENT
• Software and hardware for the internal activities; IP and server acquisition.
Primary Activities
INBOUND LOGISTICS
• Client data;
• Contacts lists;
• Campaigns data;
OPERATIONS
• Develop, maintain and improve E-Goi products;
• Logistic and social marketing integrations;
• Advanced marketing reports.
OUTBOUND LOGISTICS
• Campaigns multi-channel;
• Statistical reports about campaigns.
MARKETING SALES
• Customer success;
• Blog and E-Goi community;
• Customer acquisition and churn improvement;
SERVICE
• Help-desk;
• Assets management;
• Knowledge-base;
Margin
Margin

Figure 4.3: E-goi value chain diagram

Value Proposition

A value proposition can be perceived as an explicit promise made by an organization to deliver a particular aggregation of value-creating benefits. To put it another way, a value proposition is an organization's focus on elements that create differentiation when it comes to the customer's decision to purchase our products or services instead of the competition's [43].

Using a Business Model Canvas, one can capture the value proposition of the proposed solution in the E-goi ecosystem. This canvas, originally developed by Alexander Osterwalder, consists of a framework that ensures that there is a fit between the product and the market - customer segments and value propositions. This model is often used when there is a need to refine an existing product or service or when a new product is being developed or in stage to be developed [44].

In this case of the solution being developed, the customers can be both the final users of the E-goi platform, the working team members of the development team, since they are the direct affected users of the platform and ultimately the remaining collaborators of E-goi since they can also submit new issue reports. In the Figure 4.4, is presented the value proposition for the Issue Creator platform and the characteristics it possesses in order to create value for the customers.

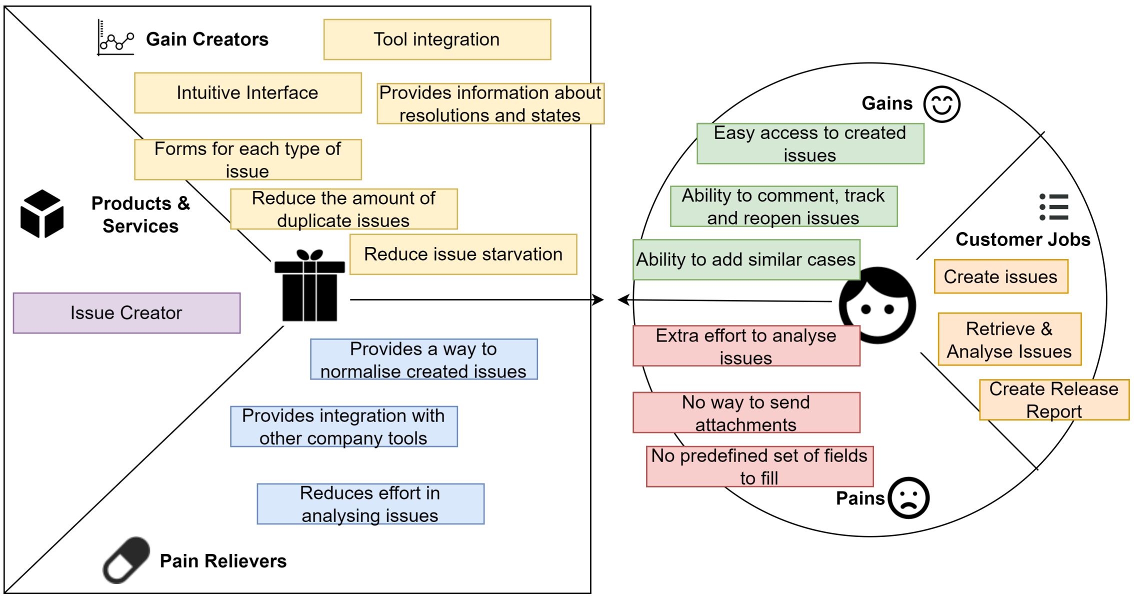

Gain Creators
Tool integration
Intuitive Interface
Provides information about resolutions and states
Forms for each type of issue
Products & Services
Reduce the amount of duplicate issues
Reduce issue starvation
Issue Creator
Provides a way to normalise created issues
Provides integration with other company tools
Reduces effort in analysing issues
Pain Relievers
Gains
Easy access to created issues
Ability to comment, track and reopen issues
Ability to add similar cases
Customer Jobs
Create issues
Retrieve & Analyse Issues
Create Release Report
Extra effort to analyse issues
No way to send attachments
No predefined set of fields to fill
Pains

Figure 4.4: Product value proposition

The value proposition canvas is divided into two building blocks. On the left hand side is the customer profile with the following elements:

- **Gains**: the benefits which the customer expects and wants, what elements would increase the odds of adopting the value proposition;
- **Pains**: the negative experiences, risks and emotions that the customer has when trying to get the job done;
- **Customer jobs**: the functional, social and emotional tasks that the customers try to perform.

On the right hand side, there is the company's value proposition, with the following elements:

- **Gain creators**: how the product or service creates customer advantage;
- **Pain relievers**: how the product or service alleviates the customer's pains;
- **Products and services**: the products and services which will create gain and are being explored.

4.2 Requirements Engineering

Requirements engineering is most commonly seen as the first phase in the software engineering process and is considered a key task in software development. In fact, a well-implemented requirements engineering reflects in the software development process in the measure that it improves the productivity and quality of the product [45].

Concerning the requirements, these can be categorized into two types: functional and non-functional. Functional requirements can be understood as requirements that specify an action that a system must be able to perform, without considering any physical constraints. It must specify the input/output of a system [46].

On the other side, non-functional requirements can be described as the constraints in which the system must operate and the standards which must be met by the delivered system such as delivery time or technical constraints like programming language to be used. As opposed to functional requirements, non-functional requirements can be difficult to gather, define and even prioritize, since there is no complete list and no standardization on a framework or taxonomy to support them [47].

However, the quality of the developed software can be appraised through non-functional requirements, which, in turn, necessitates a way to identify the most significant software characteristics for a piece of software [47]. Thus, several quality models have been developed and published. One of those models is the FURPS quality model, which was first introduced by Robert Grady at Hewlett-Packard in 1992 and later improved to be known as FURPS+ by IBM. FURPS relates to functionality, usability, reliability, performance, and supportability and the extension (+) allows the specification of constraints such as design, implementation, interface, and physical constraints.

In this section, elicitation and gathering of functional and non-functional requirements are described. In the functional requirements subsection, the list of all gathered requirements is presented and in the non-functional requirements section is presented a list of quality attributed segmented accordingly to the FURPS+ model.

4.2.1 Functional Requirements

According to Sommerville [48], the requirements engineering phase consists of four main activities, namely: discovery through elicitation, analysis and negotiation, documentation, and, finally, validation. In the discovery phase, requirements are gathered which consist of the essential characteristics of a system and conditions related to the development process.

Elicitation has the main objective of discovering the purpose of the system under development. In this phase, requirements are discovered through the application of elicitation techniques. Even though dozen of techniques are available, only a few are thoroughly explored and proven to work effectively [45]. Midst these techniques are the ones used in the requirements gathering for the current project. These techniques consisted of **traditional techniques**, through interviews and meetings with the CTO and **group elicitation techniques** which were conducted with the UXA department in order to gather the requirements needed for the forms used in the educational content related issues. This last technique is useful in the sense that it allows stakeholders to communicate requirements more willingly through brainstorming ideas.

When putting into practice the elicitation techniques described above, the following requirements were gathered and which are described in the Figure 4.5.

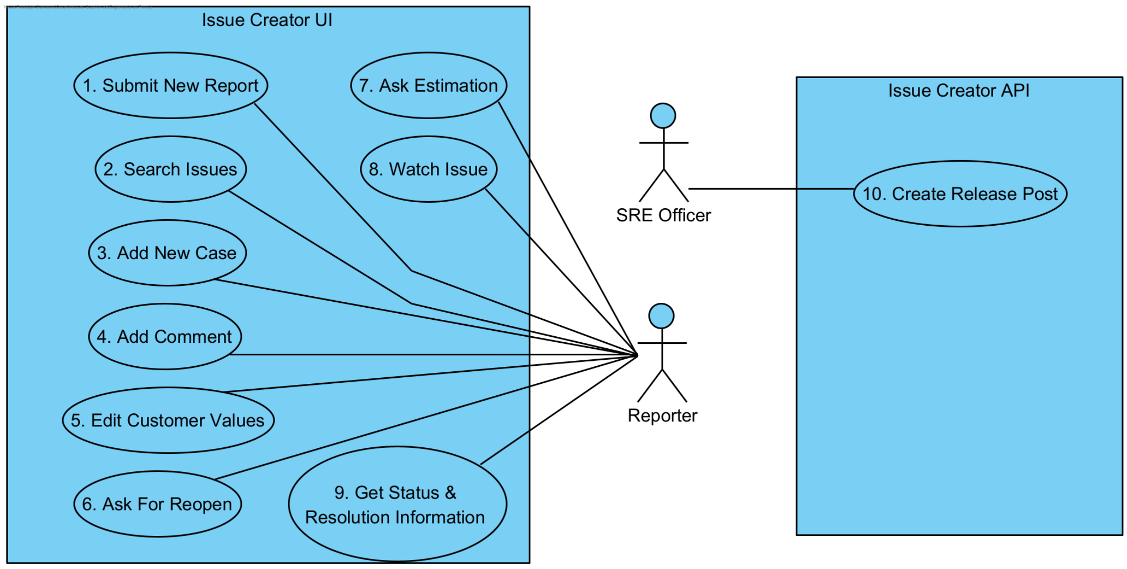


Figure 4.5: Use Cases diagram

As observable in the previous figure, two actors were identified during the requirements gathering phase:

- **Reporter**: consists in the user that reports the issues, it can be any collaborator inside E-goi that has permissions to do so;
- **SRE Officer**: represents the Site Reliability Engineering department officer, which has, among other responsibilities, the creation of the release post with all the information related to the issue corrections that were made in the current release.

After the requirements gathering with the CTO and prior analysis, it was possible to create a list of functional requirements to be included in the final solution. These requirements will be described in the format of user stories for each one of them in the Table 4.8.

Table 4.8: Functional requirements list resultant from elicitation

Functional Requirement	**Description**	**Actor**
FR01-Submit New Report	As a reporter, I want to be able to create a new issue report. I want to be able to fill in the necessary data for the type of issue I want to report.	Reporter

Functional Requirement	**Description**	**Actor**
FR02-Search Issues	As a reporter, I want to be able to search for issues that are not only reported by me but also all the available issues that aren't in a "Done" state.	Reporter
RF03-Add New Case	As a reporter, I want to be able to add a new case to an existing issue. A new case consists of a similar issue that occurred or that was reported by a client and that is related to an already created issue. This action should increase the visibility of the issue to the managers.	Reporter
FR04-Add Comment	As a reporter, I want to be able to add a new comment to an issue plus add any attachments that are deemed to be relevant to the comment.	Reporter
FR05-Edit Customer Values	As a reporter, I want to be able to edit the customer values referent to a specific issue in order to update or correct eventual mistakes to these fields made during issue reporting. These fields consist on **Business Value**, **Customer Impact** and **Customer Satisfaction**.	Reporter
FR06-Ask For Reopen	As a reporter, I want to be able to ask to reopen an issue that, after further analysis/tests or client feedback, was deemed to not be properly resolved. This action should increase the visibility of the issue to the managers.	Reporter
FR07-Ask Estimation	As a reporter, I want to be able to ask for an estimation on the average time for the issue to be resolved. This will later be responded by the issue assignee, product manager or CTO. This action should increase the visibility of the issue to the managers.	Reporter
FR08-Watch Issue	As a reporter, I want to be able to watch an issue, which means I want to be able to receive notification whenever the issue changes status or resolution, basically whenever an issue is updated.	Reporter
FR09-Get Status & Resolution Information	As a reporter, I want to be able to see the information related to issue status (description, current owner/responsible and maximum time in the correspondent status) and issue resolutions (description)	Reporter

Functional Requirement	**Description**	**Actor**
FR10-Create Release Post	As a SRE Officer, I want to be able to create a release post in Bitrix with all the issues that are currently in for the new release. The information to be included must be enough to identify the issues, what they resolved, and how they were resolved.	SRE Officer

4.2.2 Non-Functional Requirements

The relation between functional and non-functional requirements is that the latter should reflect how the system should behave, unlike functional requirements which reflect what the system should do.

In order to capture these requirements, the FURPS+ model was used. This model takes into account various vectors of non-functional requirements for better categorization. These categories are composed by: Functionality, Usability, Reliability, Performance, Supportability and the extension (+).

According to the acquired elements only a few was captured. Regarding the usability, the user interface must be easy to use, in order to be used by low experienced tech users. On the other hand in the Supportability, the application must be prepared to include new types of reports or functionalities with minimal impact on already developed features. In the same way, it must support integration with other external systems (Bitrix24 and LiveAgent) and must log the request in order to detect possible errors in processing. On the other hand, in what concerns the extension, the following items were retrieved:

- **Design Constraints**: Must follow good practices in software development such as the adoption of the SOLID [1] and General Responsibility Assignment Software Patterns (GRASP) patterns;
- **Implementation Constraints**: Must be developed using technology present in the company tech stack.

4.3 Summary

In this chapter both a research on the final value of the solution and the presentation of the elicited requirements was conducted. Regarding the value analysis, by applying the methods presented enabled one to conclude that, from all the various solutions, the most appropriate would be to create a custom-made solution that could aggregate the various systems and provide the means for the users to have an easy access to the created issues, the ability to add similar cases and thus, enable the use of buzz-points to aid in the prioritization system and the ability to create standardized issue reports.

In the next section, is presented the design of the envisioned solution which consists in the next logical step to perform in order to create a robust solution that can take into account the elicited functional and non-functional requirements.

[1] Created acronym that relates to the five principles of object oriented programming

Chapter 5

Design

This chapter's objective is to provide a clear view of the overall system architecture, taking into account the requirements gathered previously. The Unified Modeling Language (UML) diagrams presented in this chapter allow a deeper understanding of the main relationships between the components, as well as all the decisions made during software development.

5.1 Components

The component diagram allows the designer to show the relationship between the different components in a system. It is a module of classes that represent independent systems that have the ability to interface with the rest of the system. A generalized view of the components' interactions is presented and described in this section. The components diagram for the final solution is presented in Figure 5.1.

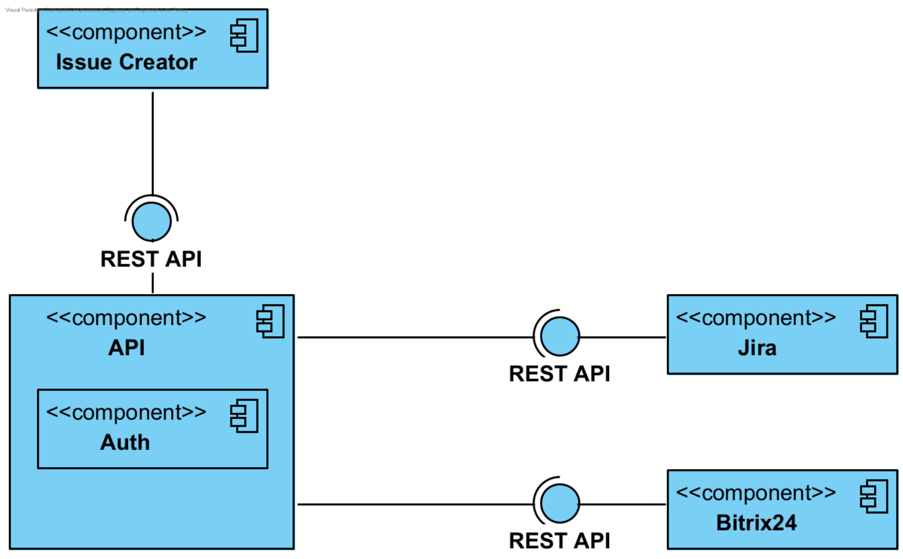


Figure 5.1: Components diagram

According to the figure, the component of the **Issue Creator**, which presents the user interface to be accessed by the browser, consumes via a REST API the API used to format the requests, provide business rules and forward the requests to the other services integrated within the solution. The API, on the other hand, provides an internal component responsible for the authentication in the external services that it consumes - Jira and Bitrix24.

An API provides a secure way to access data, enables data sharing homogeneity in the requests, and facilitates the integration of different platforms that can provide access to resources using various protocols, so it becomes necessary to implement an API to separate the user interface from the remaining platforms. In the same way, the use of an API also offers a way to implement a solution envisioning the implementation of the Separation of Concerns principle, by passing the responsibility of dealing with authentication and authorization and data formatting to a single component. Thus, the application responsible for the presentation layer is segregated into another component - the **Issue Creator** component.

5.2 Deployment

Regarding deployment, a deployment diagram provides a way to visualize the topology of the physical components of a system, how and where the software components are deployed. Regarding the project deployment, it is described in the Figure 5.2.

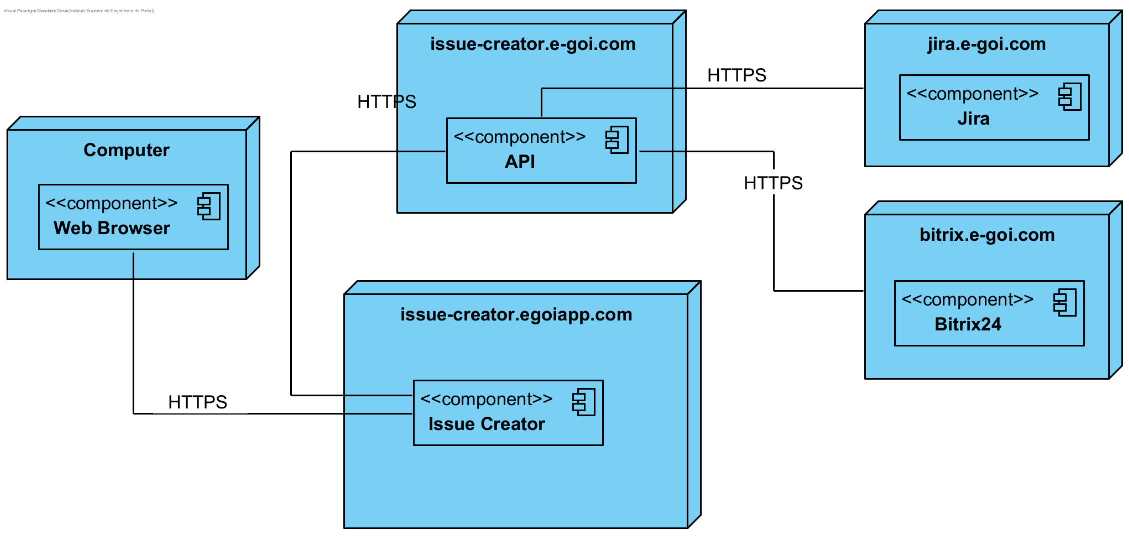


Figure 5.2: Deployment diagram

According to the figure, the physical devices that constitute the system are as follows:

- **Computer**: corresponds to the client physical device that uses a browser in order to access the platform;
- **issue-creator.e-goi.com**: domain responsible for the deployment API;
- **issue-creator.egoiapp.com**: the domain responsible for the deployment of the web application - the Issue Creator component;
- **jira.-egoi.com**: the domain responsible for the deployment of the Jira platform;
- **bitrix.e-goi.com**: the domain responsible for the deployment of the Bitrix24 platform.

It is relevant to point out that all the domains represented were already implemented. In this case, the only domain that was newly created was the one where the web application should reside.

5.3 Use Cases

With the architecture defined, it becomes possible to analyze all the requirements in order to specify how they are to be implemented. This will enable one to specify how they should work and for what. In this section, a description of the design and constraints of each one of the requirements is described and the flow of the messages between the components is presented with the use of sequence diagrams, designed using UML.

5.3.1 FR01-Submit New Report

This requirement relates to the submission of a new report of a determinate issue type. The sequence of actions to be performed by the actor (reporter) is presented in Figure 5.3.

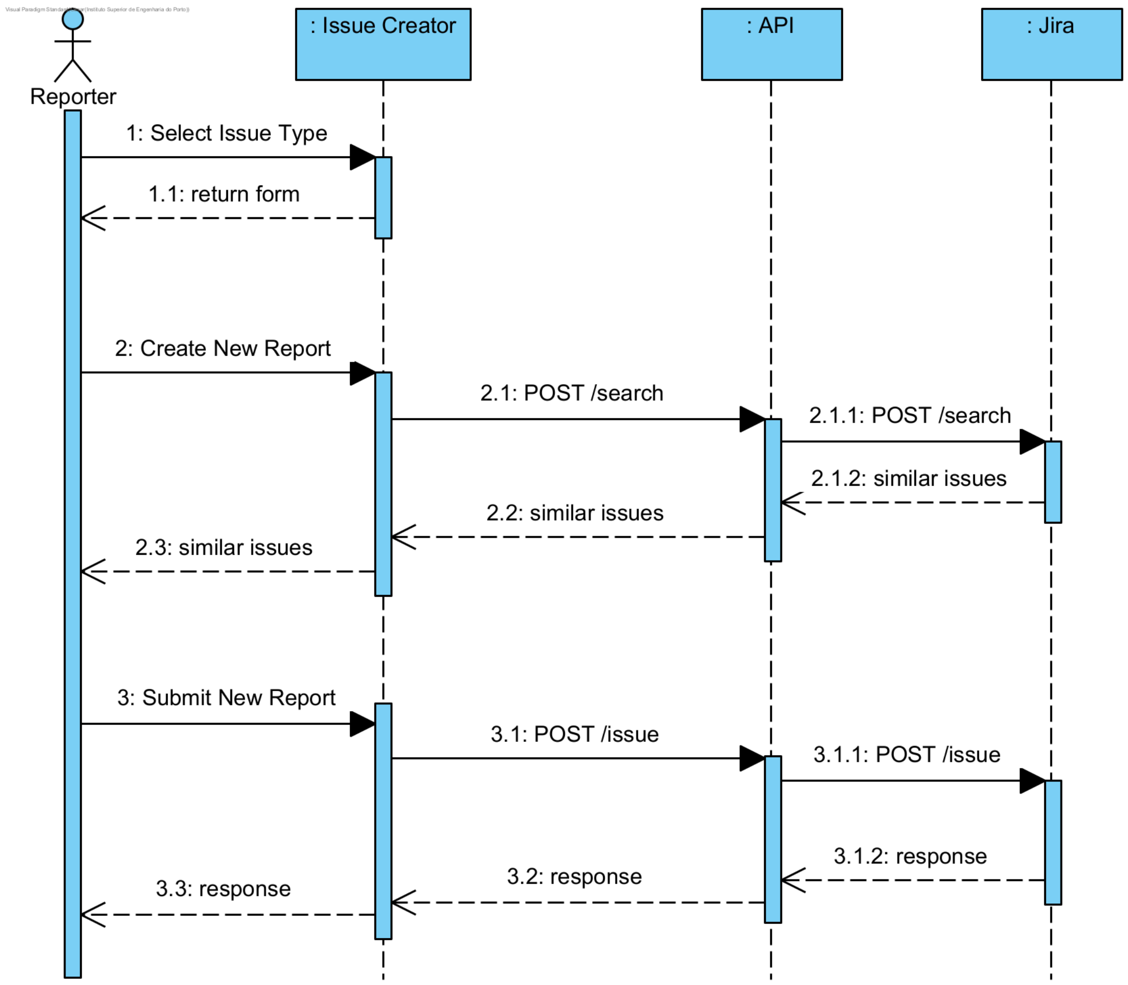


Figure 5.3: Submit new report sequence diagram

The reporter begins the requirements by selecting the type of issue that he wants to submit. After selecting the required issue type from the four available - bug, feature, improvement or educational (more information below), the reporter can fill in the fields of the form and ultimately submit the report.

In response to a stakeholder request, before submission (during the writing of the report), to aid in the detection of issue duplications, Continuous Querying can be implemented as described in the literature review section (Section 3.2). When the user inputs the summary (short description of the issue), after a short delay, the API can be queried to retrieve similar issues that contain similar statements in both the description and the summary. It then falls to the responsibility of the reporter of the issue report to verify each one of the retrieved issues (10 maximum) and check if it has already been reported. In case an already reported issue is found, the user should instead open a new case for that issue - requirement FR003.

After everything is filled out properly, the reporter submits the issue, in which case, a request is sent to the API and then to the Jira tool via REST. The URL paths demonstrated in the picture correspond to actual paths to the Jira REST platform.

Regarding the information that should be presented to the user for any of the issue types, these must follow the rules presented in the Section 3.1. Bellow is a description of all of the issue report types that must be implemented. Regarding the educational type, the contents of this report are in accordance with the specifications gathered when interviewing the UXA team.

Bug

A bug represents a problem that impairs or prevents the functions of the product. In the case of bug reporting, the information that must be provided by the reporter must be straightforward and complete. This is in order to help the developer to understand the problem at hand. Thus, the information that must be included in the form is:

- **Summary**: a brief description of the issue. Must be direct and concise to describe the issue. This field is mandatory;
- **Component**: Relates to the software component responsible for that issue. This is used to automatically assign the issue to the corresponding team manager for review. Jira handles this internally since the users have already been assigned to each component. This field is mandatory;
- **Client Information**: corresponds to additional information regarding the information gathered from the reporter client. This includes fields such as client ID, support ticket ID, campaign ID, etc. This information is optional but necessary;
- **Current Behavior**: the current behavior of the action performed. This field is mandatory;
- **Supposed Behavior**: what was supposed to happen when performing the use case. This field is mandatory;
- **Steps to Reproduce**: what actions can be taken to reproduce the issue. This field is mandatory;
- **Workaround**: what steps were taken in order to solve the problem temporally;
- **Additional Information**: any additional information that can help to analyze the problem, this includes attachments;
- **Customer Values**: values used in order to apply the MoSCoW prioritization method (described in Subsection 2.4.2);

New Feature

A new feature is a component or feature of the product that has not yet been developed. The fields to be included in this section are similar to the ones present in the bug report but instead of the supposed behavior, actual behavior, steps to reproduce and workaround, two mandatory fields are implemented: desired behavior (what should be implemented) and goal (why it should be implemented).

Improvement

An improvement is an enhancement to an existing feature or task. Similar to the new feature report, this report should include three different fields: current behavior, desired behavior and problems to solve.

Educational

The educational content issue type reports correspond to an update or creation of an article in the Knowledge Base that needs to be performed by the UXA team.

According to the user feedback gathered in the presence of the UXA team, this form must take information according to the four types of requirements of the team. This can be achieved with a dynamic form that is generated when choosing from a number of alternatives such as knowledge base information update, new feature documentation, customer doubts and other requirements.

An update of information must include a link to the knowledge base, the test account, the outdated information, and the information to update. In the case of new feature documentation, the fields **test account**, **feature description** and **feature flow** must be described. There are two fields in the customer doubts form: ticket ID and client questions. This form is used in order to identify which information seems to be unclear or missing in the knowledge base so that it can be further investigated. Lastly, the other cases need only a simple issue description.

For all the issue types and in all the fields, a short description of what and how to write must be presented to the user. This will guide the filling of the different fields and prevent misinformation. This includes a short statement in the summary of the good and bad examples of issue summaries.

5.3.2 FR02-Search Issues

The objective of this requirement is to search issues in order to be able to easily access the other functionalities related to the prior selection of the issue to act upon - FR03 to FR08. A sequence diagram for this action is shown in the Figure 5.4.

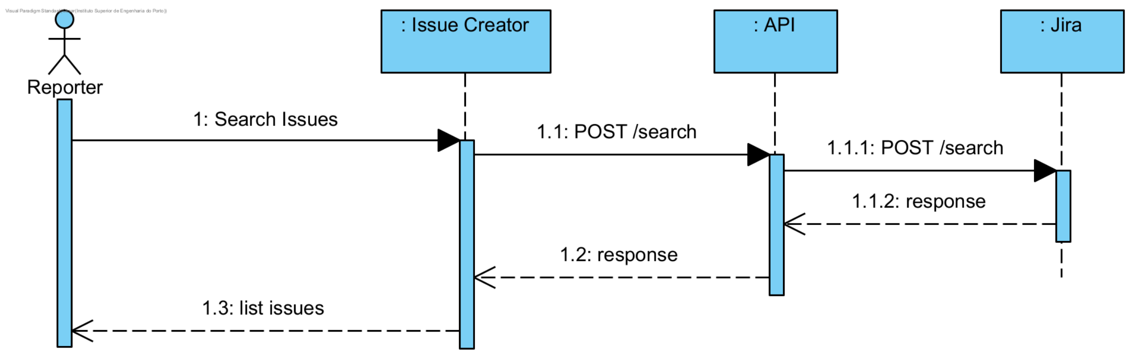


Figure 5.4: Search issues sequence diagram

The user interaction with the platform is very simple and straightforward. The user starts by querying the issue that he wants to find. The request is sent to the API and forwarded to the Jira platform trough REST, the results are then shown to the user where he can visualize the information and perform the various actions. In order to improve the search functionality, filtering options are to be implemented to filter issue types and to show the reporter submitted issues.

Jira REST API search feature allows both the GET and POST methods, however, according to their documentation, the later one is more efficient and allows and more comprehensible understanding and organization on the sent parameters. The queries performed to retrieve the issues from Jira use the language Jira Query Language (JQL), which represents the query language used by Jira to make searches for issues.

5.3.3 FR03-Add New Case

The objective of this requirement is to allow the reporter to report an issue that has already been reported but that has also been experienced by another customer. This is done through feedback. Figure 5.5 represents the action sequence diagram for this requirement.

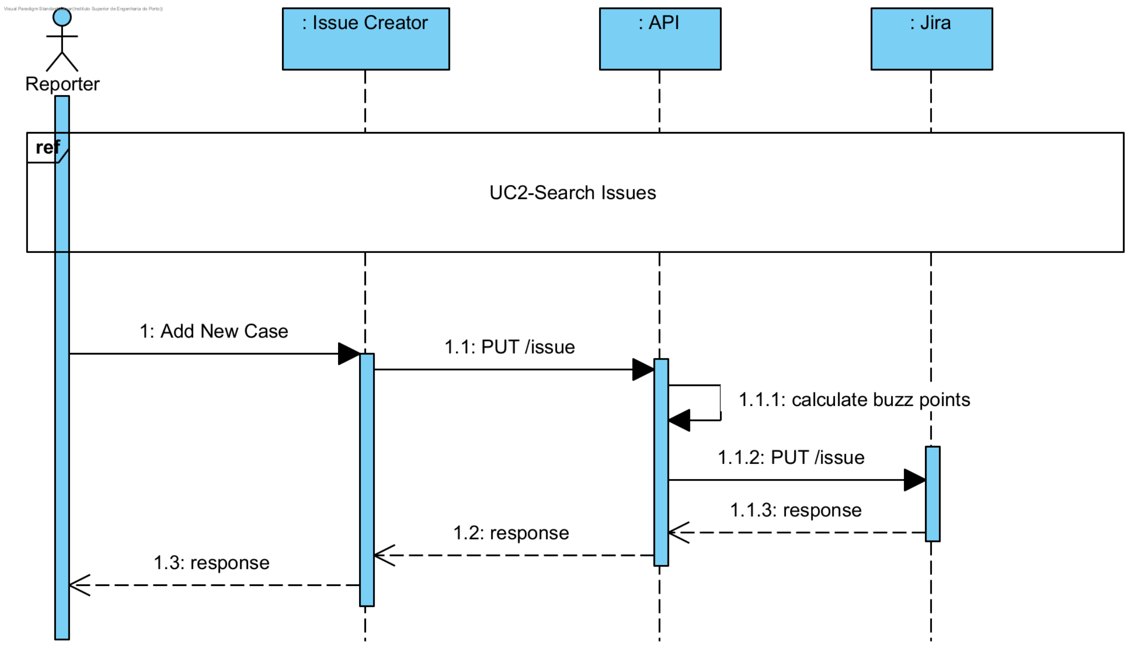


Figure 5.5: Add new case, sequence diagram

According to the figure, the reporter, when adding a new case, fills in the mandatory fields and submits the request. The request is then sent to the API, where the buzz points are calculated and the request is forwarded to Jira. In this case the PUT method is used because Jira allows the attachment of comments when updating the issues information. This allows the appending of a new comment with the information regarding the description provided by the reporter.

A new case can be submitted by providing a description, attachments, and most importantly, the customer values. By adding a new case, the reporter presents the new customer values for that case found and, if the values are superior (more relevant) than the ones originally in the issue, then these values must be updated with an attached note.

Regarding the constraint of increasing the visibility of the issue, the term **Buzz Points** comes into play. This is an integer value that is incremented whenever an action is required to make an issue stand out from the rest. This requirement, along with FR06 and FR07, should increase this value.

The addition of the Buzz Points allows an issue to have a bigger priority than others that are in the same MoSCoW priority range. This gives them more visibility to the managers that are responsible for analyzing them.

5.3.4 FR04-Add Comment

This requirement allows the reporter to add a new comment to an existing issue. The addition of comments is used as a means of communication between the reporter and the current issue assignee to add extra information or to discuss other matters.

The fields that must be present in this form are the comment which is mandatory and additional information which is optional and also includes the ability to send attachments. The sequence flow of this requirement is similar to that of adding a new case (FR03).

5.3.5 FR05-Edit Customer Values

The aim of this requirement is to allow the reporter to edit the customer values due to an error committed when submitting the issue or simply to increase the priority of the issue. According to the data, in addition to editing the values, the reporter must also add a justification for altering them.

Table 5.1, describes all the values that the customer values can take and their priorities. The priorities are evaluated from 0 to 5, with zero being the minimum priority.

Table 5.1: Customer Values description

Group	**Value**	**Priority**
Business Value	Unknown / NA	0
	Trial / Free	1
	Starter	2
	Pro / Pay as you go / Internal Customer	3
	Corporate	4
	Top Corporate	5
Customer Impact	Unknown / NA	0
	It does not have a big impact on the client's tasks in E-goi. Nice to have	1
	It has some impact, but there is a simple workaround	2

Group	Value	Priority
	The impact is medium, causing some discomfort to the customer	3
	High impact, the workaround for client activity in E-goi is complex	4
	It is a blocker for client activity in E-goi	5
Customer Satisfaction	Unknown / NA	0
	Very Satisfied	1
	Satisfied	2
	Indifferent	3
	Lack of Confidence	4
	Churn Imminence / Conversion Loss	5

5.3.6 FR06-Ask For Reopen

Asking for a reopen of an issue occurs when the reporter decides, upon extensive analysis, that the issue that was once closed with a specific resolution was not properly implemented and the resolution was not satisfactory. As a consequence, the reporter can ask for a reopening of the issue, which would consist of increasing the Buzz Points and adding a new comment.

The sequence of the actions and transactions between components is similar to Figure 5.5.

5.3.7 FR07-Ask Estimation

Using the ask estimation action, the reporter can add a new comment to an issue to request an estimate of resolution time for developing or closing the issue.

This requirement takes no fields as it just adds a comment to the issue with the name of the reporter asking for an estimation. In essence, this action follows the same steps as FR03 and should add up to the current Buzz Points of the issue.

5.3.8 FR08-Watch Issue

The goal of this requirement is to allow the reporter to watch an issue. This allows him to receive notifications whenever an issue is updated, whether it be by adding a new comment or status and resolution changes. This also facilitates the reporter in finding issues when using the search feature.

Similar to the requirement FR07, this action takes no fields as it is a direct request to the Jira API trough the watchers route, passing the id of the issue and the reporter id.

5.3.9 FR09-Get Status & Resolution Information

The objective of this requirement is to provide reporters with information about the status and resolution of issues and their meaning - Table 2.2 and Table 2.3 in Subsection 2.5.2, respectively.

The information regarding the items (description, title, owner and maximum time) is retrieved dynamically from the Jira component and then formatted and returned by the API. The routes used to retrieve the information from Jira are "status" and "resolution".

5.3.10 FR10-Create Release Post

When creating the new product release, the SRE Officer has the need to create a post in Bitrix24 in order to announce a new product version to the entire company, along with the issues that were resolved along with their respective information such as issue ID, issue type, component, summary, resolution and priority score. This action is to be triggered from GitLab or through a terminal, which means that a route for this action is to be developed in the API in order to integrate into the GitLab Webhooks.

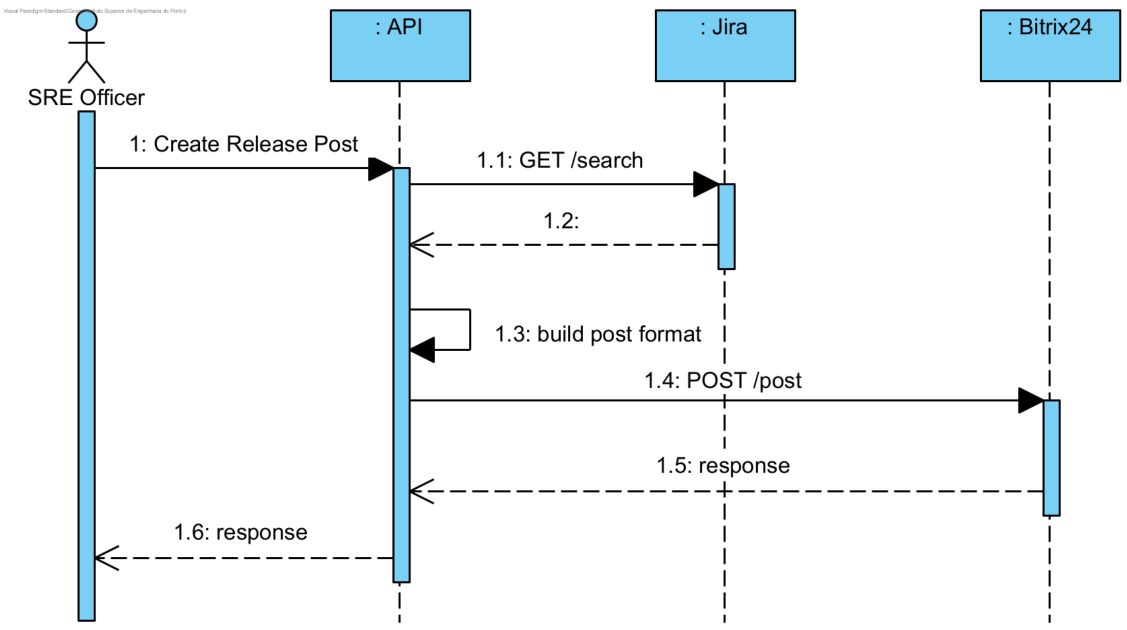


Figure 5.6: Create release post sequence diagram

In order to create a release post, the SRE Officer must provide the release version. In the API, the issues that concern that version are retrieved. These issues then need to be formatted in order to create the post message and to match the Bitrix24 required request data in the POST method. After the formatting, the request is sent to the Bitrix24 component and the response is retrieved.

5.4 Alternative

Despite the fact that the solution chosen to prevent duplicate issues serves its purpose and meets the stakeholders' requirements. In terms of performance this might not be the best applicable scenario. In fact, this is one of the main constraints of the chosen algorithm as stated in [16].

An alternative that could in fact be applied to solve the performance issue would be to use a duplicate prevention approach. This means detecting the duplicate after submitting the issue. The algorithm applied could be the same, however the issue submitted would be processed in a later phase. However, one thing to consider here was that the user would most likely not be more resultant to search for similar issues in order to submit a new case instead of simply creating a new issue report.

For this alternative, Figure 5.7 presents the sequence diagram of overall envisioned solution. At first all the issues present in Jira would have to be indexed in Elastic Search, an open-source search engine that provides a REST API and the scalability and performance for indexing and retrieving unstructured documents almost in real-time [49]. This engine would be used in order to not be dependent on Jira limitations when querying for all the issues.

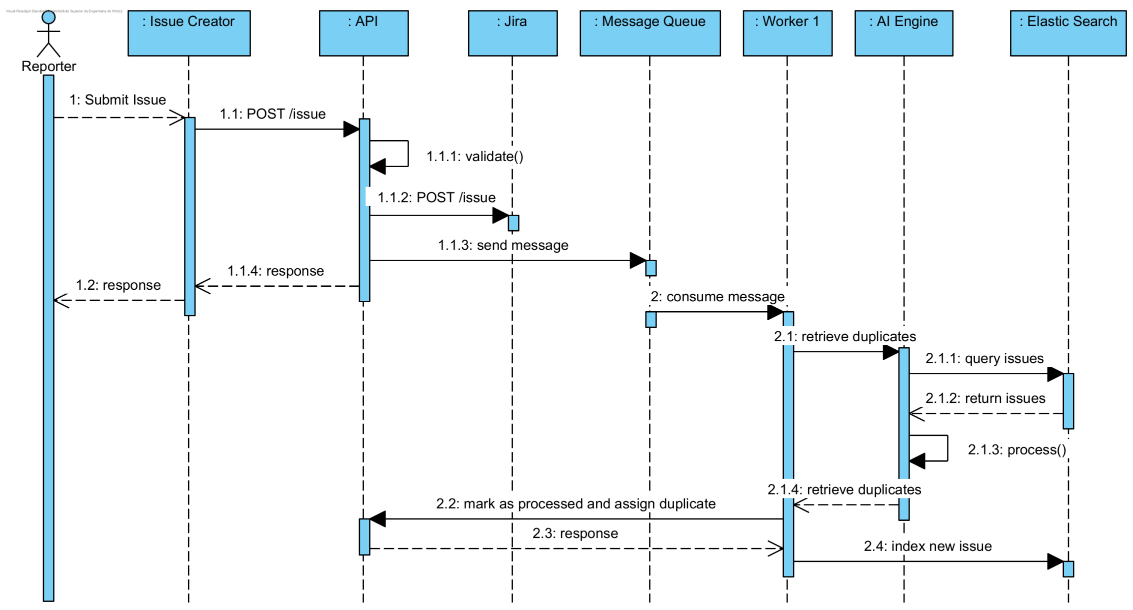


Figure 5.7: Alternative solution sequence diagram

According to the figure, the reporter submits a new issue and the request is passed on to the API. The data is validated and the issue is created in Jira with a new state for example - **Unprocessed**.

Next, a new message is sent to the message queue, an engine that allows for asynchronous service-to-service communication, in which messages are stored until they are processed by a consumer (or worker). This method helps to reduce the workload by enabling the scaling of more workers if needed in order to decouple heavyweight processing.

After that, the worker consumes the message and starts processing it. In a general overview, what it does is communicate with the Artificial Intelligence (AI) engine. This engine, in turn communicates with the Elastic Search engine in order to retrieve the issues to compare to. The AI engine processes the issues, finds the top-n duplicates in order of matching percentage and passes the information to the worker. Here, the next step is to check if the percentage of the first matching issue is high enough (for example above 90%). If the retrieved issue passes the condition then a request is sent to the Jira platform in order to pass the issue to a **Processed** state, link the found duplicate to the issue and add the label **Duplicate**.

After this entire process, the worker must finally create a new indexation on the Elastic Search engine in order to include the newly created issue.

Even though this solution could bring better performance and scalability for the envisioned application, the fact is it would not reflect the requirements of the stakeholder. This solution however could be implemented in a later phase when considering enabling the E-goi clients to submit their own issues, since the volume of data would increase and a more consistent deduplication solution would have to exist. Due to the fact that the organization already uses these engines, it would only be a matter of implementing communication between them.

5.5 Summary

In this chapter, was presented the design of the final solution taking into account all of the requirements. Here, were presented all the technical aspects of the system (architecture, communication, deployment, and functional requirements logical flow). Additionally, alternative solutions were included, in case performance becomes an issue for further application scalability.

Next, comes the implementation of the system, taking into consideration all the planning performed in the present chapter.

Chapter 6

Implementation

After the study, planning and analysis phases, the implementation phase of the idealized solution is now performed. In this chapter, the most relevant elements of the implementation of the Issue Creator will be presented.

This chapter presents the technologies used, the development methodology and the implementation of the front-end and back-end (API) of the platform.

6.1 Technological Environment

For the development of this project, the Angular [1] framework was used for the development of the front-end and the Python [2] language with the Falcon framework [3] and Gensim [4] for the development of the REST API (back-end).

For the development of the two types of tests in the Angular project (unit and integration), other two frameworks were used. These will be mentioned in Subsection 7.1.1. In the next paragraphs each of these technologies will be briefly mentioned. It is important to mention that both of these technologies were used since they are part of the technological stack of E-goi.

Angular

Angular consists of an open-source, front-end framework developed by Google that aims to provide an improved typed version (through Typescript) of its predecessor - AngularJS.

This platform includes various advantages such as component-based development that foments re-usability and maintainability, out-of-the-box tools that allow routing management, external communication and forms management, and a set of features that facilitate the development and testing [50].

Python

Python is a high-level, object-oriented, modular, open-source programming language that supports not only rapid learning thanks to a straightforward syntax, but also allows for fast development and maintenance of code in large projects.

[1] https://angular.io
[2] https://www.python.org
[3] https://falcon.readthedocs.io
[4] https://radimrehurek.com/gensim

One of its most attractive aspects is the fact that it provides a vast number of native and third-party libraries, making it very popular and useful in a large multitude of sectors, such as web development, data analysis and machine learning.

Regarding the most significant frameworks used in the project, we can highlight two of them - Falcon, that allows the creation of REST API's and microservices with a major focus on reliability and performance [51], and Gensim, that corresponds to a library that allows the processing of various documents and their comparison through their prior transformation into vectors and subsequent application of machine learning algorithms [52].

Regarding the Gensim library, in order to be able to use it, some core concepts must be understood such as:

- **Document** - represents the text we want to parse in order to create the vectors and train the model;
- **Corpus** - represents the collection of all the documents submitted for training the topic model;
- **Vector** - representation of each Corpus in a structure that enables it to be manipulated mathematically;
- **Model** - refers to a transformation between one document representation to another.

6.2 Development Methodology

Despite the project being carried out by just one person (in general), in order to develop this project, the adoption of the development methodologies implemented by E-goi took place. This methodology allows the various teams to work in an environment that fosters a continuous integration continuous delivery approach.

In this regard, continuous integration corresponds to a widely established practice in which the team members integrate and merge developed code frequently and continuous delivery corresponds aims at ensuring that an project is constantly production-ready by running automated actions every time a project is merged [53].

E-goi uses the platform GitLab [5] as a Git tool for version control for all its projects. This tool allows the integration with other platforms such as continuous integration tools.

Both for Issue Creator and for the other projects present in E-goi, in terms of versioning, there are three branches - one for development, one for carrying out the different tests and the last one for production. When developing a new functionality, the development process must pass through the three branches.

In order to automate the process of continuous integration and make available any functionality developed for the end user, or, in other words, in order to automate the process of code verification, test execution, project building and deployment for both the test and production environments, the Jenkins tool is used [6]. This tool is integrated with GitLab and allows the management of continuous integration pipelines. Whenever new code is sent to the test branch, the configured pipeline for the project (and that branch) is triggered and the various actions are performed.

[5]https://about.gitlab.com
[6]https://www.jenkins.io

The pipeline for the Issue Creator project contains the following actions - branch checkout, project build, test execution, verification of the code, and, finally, deployment to staging. Regarding the production pipeline, the process is the same but, in this case, the code can only be sent to the master branch once the feature merge request is approved by the SRE team.

Applying this methodology will ensure that there will be a significant improvement in the management of developed applications as it will ensure that the release of a new version of the application will pass through all the necessary checks to detect any bugs or improper programming practices.

6.3 Front-End Development

As explicit in the Section 6.1, the front-end application was built using Angular. The web application was developed with an eye toward maintaining its structure and the easy addition and removal of any of the functionalities or types of reports. In terms of component-based structure, a component was created for all the report types. Even though it might not be the preferred practice in terms of modularity, it compensates in terms of project structure management and code readability. On the other hand, in terms of core domain classes, these were developed by using a high level of abstraction and by using the builder pattern. This pattern allows the construction of complex objects step by step. Figure 6.1 shows the class diagram for the main classes for the forms in the web application. In this case, the factory pattern could also be applied. However, the chosen pattern offers a more readable syntax and allows validation of the various steps of constructing the final form. This includes the generation of the issue description which must be generated by passing multiple form input values for the final value.

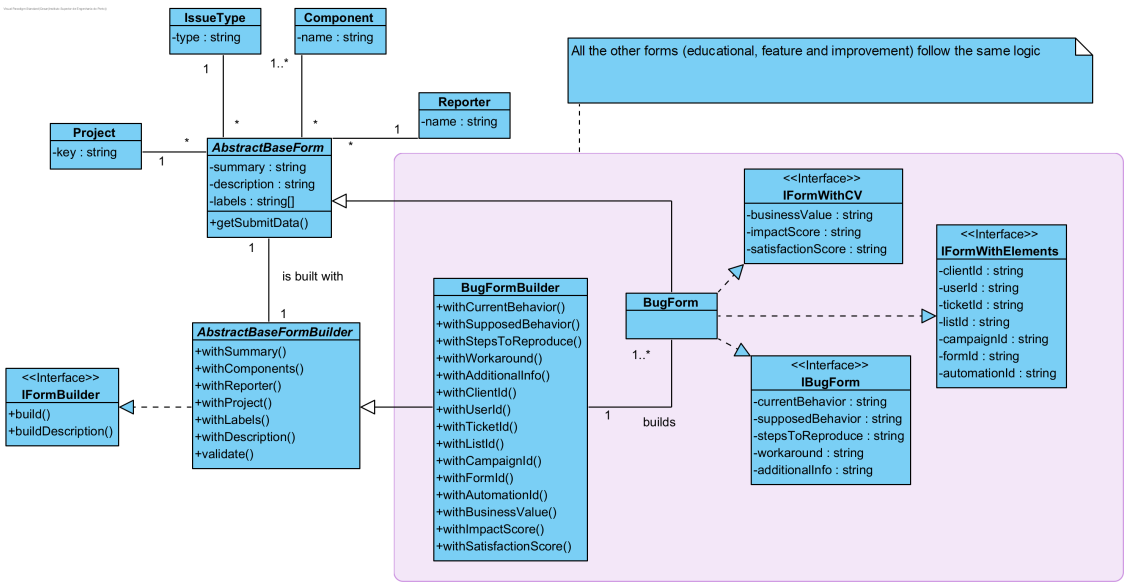
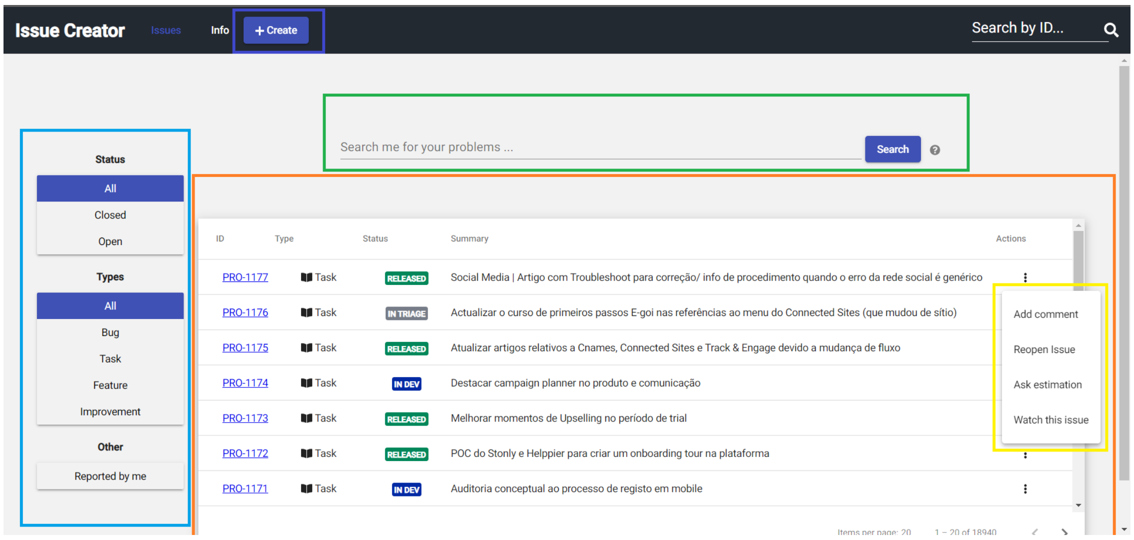


Figure 6.1: Issue Creator web app class diagram

According to Figure 6.1, each form (may it be a bug, improvement, feature or educational form) extends an **AbstractBaseForm** which consists of a summary, a description, labels, an **IssueType**, multiple **Components**, a **Reporter** and a **Project**. A form might also implement (or not depending on the fields a specific form contains) multiple interfaces such as **IFormWithCV**, **IFormWithElements** and an interface with the form-specific fields - **IBugForm**.

In the same way, each form contains a form builder, in this case a **BugFormBuilder** extends an **AbstractBaseFormBuilder**. Each builder contains a function *buildDescription()* that is responsible for building the description of the issue dynamically (in markdown format) based on the input fields that are filled at the time of submission.



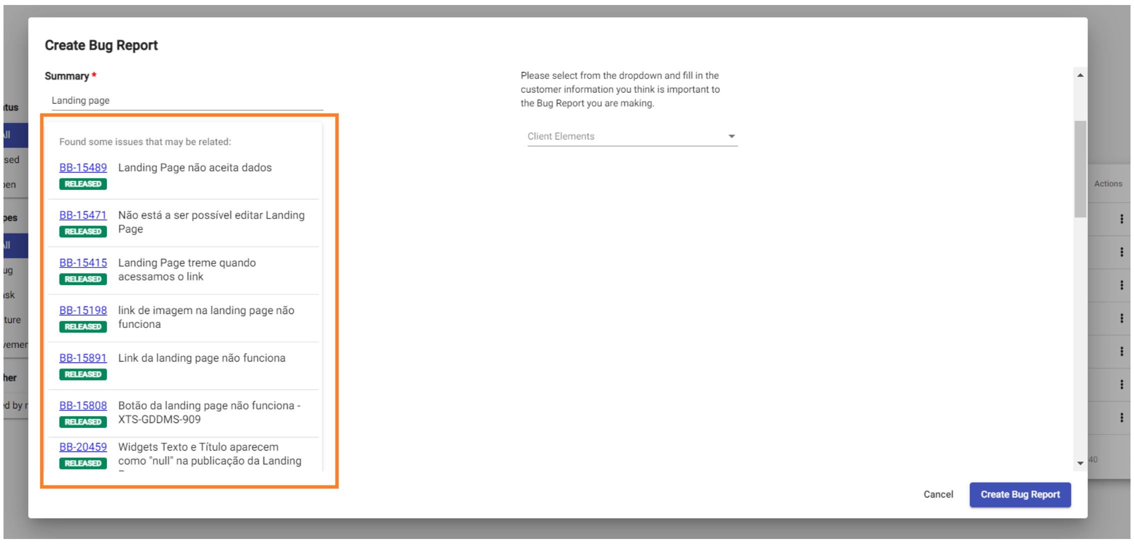
ID	Type	Status	Summary	Actions
PRO-1177	Task	RELEASED	Social Media | Artigo com Troubleshoot para correção/ info de procedimento quando o erro da rede social é genérico	⋮
PRO-1176	Task	IN TRIAGE	Actualizar o curso de primeiros passos E-goi nas referências ao menu do Connected Sites (que mudou de sítio)	
PRO-1175	Task	RELEASED	Atualizar artigos relativos a Cnames, Connected Sites e Track & Engage devido a mudança de fluxo	
PRO-1174	Task	IN DEV	Destacar campaign planner no produto e comunicação	
PRO-1173	Task	RELEASED	Melhorar momentos de Upselling no período de trial	
PRO-1172	Task	RELEASED	POC do Stonly e Helppier para criar um onboarding tour na plataforma	⋮
PRO-1171	Task	IN DEV	Auditoria conceptual ao processo de registo em mobile	⋮

Figure 6.2: Issue Creator issues search page

In terms of UI development, Figure 6.2 presents the main page of the platform. In the representation we can see the various use cases implemented by the following color scheme:

- **Yellow** - menu with the action that represents the functional requirements **FR04**, **FR05** and **FR03**;
- **Purple** - the button used to create a new issue - **FR01**;
- **Orange**, **Blue** and **green** - represents the requirement **FR02**. The lateral view allows the filtering of the various issues, with the ability to display only the issues reported by the logged reporter. The green search bar allows one to search issues by various parameters such as: issue id, summary, description, reporter, component, etc.

Regarding the functional requirement FR01, Figure 6.3 represents the presentation to the reporter of the top 10 issues that might be related to the issue being written based on the keywords provided in the summary. In this section, the reporter has the ability to check each one of the suggested issues and check if any of them is related to the one that he/she is about to submit.



Create Bug Report

Summary *

Landing page

Found some issues that may be related:

BB-15489 Landing Page não aceita dados — RELEASED

BB-15471 Não está a ser possível editar Landing Page — RELEASED

BB-15415 Landing Page treme quando acessamos o link — RELEASED

BB-15198 link de imagem na landing page não funciona — RELEASED

BB-15891 Link da landing page não funciona — RELEASED

BB-15808 Botão da landing page não funciona - XTS-GDDMS-909 — RELEASED

BB-20459 Widgets Texto e Título aparecem como "null" na publicação da Landing — RELEASED

Please select from the dropdown and fill in the customer information you think is important to the Bug Report you are making.

Figure 6.3: Issue Creator similar issues shown when filling a new report

6.4 Back-End Development

The REST API, was developed to provide a bridge between the web application and the Jira and Bitrix24 API's. As stated in Section 6.1, this API was developed in Python and uses the Falcon framework to easily serve the various endpoints consumed by the web application and the Gensim library to perform the document analysis for identification of the most similar.

In order to better comprehend the structure of this API, in the Figure 6.4 is represented a components diagram with the various components present in the application. According to the figure, the different components present are:

- **Jira Service** - responsible for the communication with the Jira REST API;
- **Bitrix Service** - responsible for the communication with the Bitrix REST API;
- **Auth** - responsible for the authentication in the Jira and Bitrix platforms. The authentication in these platforms is made by authentication token;
- **SearchBuilder** - uses the builder pattern to create and validate the query to be performed to search for the issues. This builder takes into account the various filters used to filter the issues when searching and builds the query in JQL language;
- **ContinuousQuery** - component responsible for performing the evaluation of the documents retrieved from Jira and applying the algorithms to return the top 10 most similar ones;
- **Resources** - component where the necessary endpoints reside. This component is responsible for receiving the requests and sending out the response to the web application and dispatching the webhook response.

In regard of the algorithm used to search for similar issues, the developed solution follows the technique Continuous Querying proposed by [16] (referenced in Section 3.2) in order to provide feedback to the user when filling the new report.

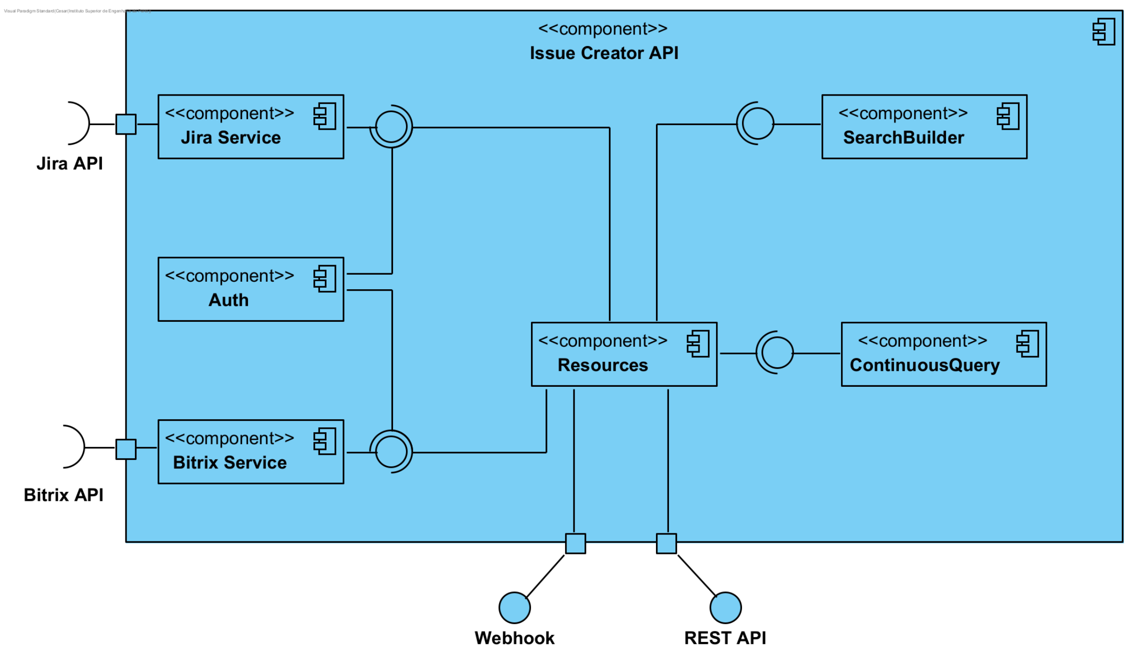


Figure 6.4: Issue Creator API component diagram

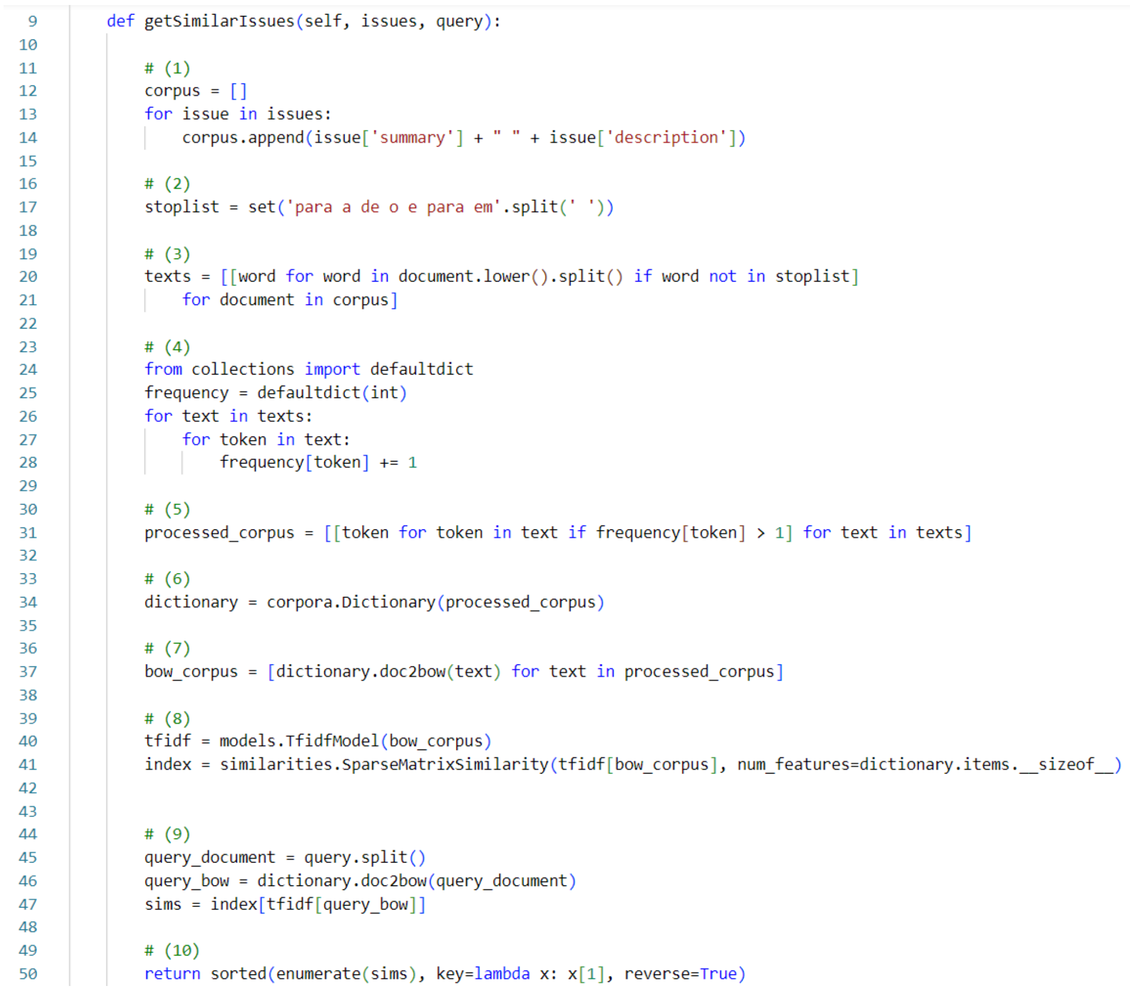
In order to implement the solution the same library and algorithm used in the article were implemented. The algorithm used was TF-IDF (Term Frequency-Inverse Document Frequency), which is the algorithm most commonly applied by search engines and consists of transforming a set of words into a vector space where the frequency counts are weighted according to the relative rarity of each work in the corpus.

In other words, *term frequency* results in calculating the number of times a term appears in a document. On the other hand terms such as the most common words ("the", "of", "a", "as", etc) cannot accurately determine the relevance of a document, thus *inverse document frequency*, is introduced in order to solve this problem. Thus, the inverse of the frequency of the terms is used in order to reduce the weight of the terms that occur more frequently. This is done by increasing the weight of the terms that occur rarely.

The library used, offers a simple and straightforward implementation of this algorithm in order to identify the top 10 articles with the highest probability of being duplicates. In Figure 6.5, is represented the code used in order to find the similarity of the queried terms against a group of retrieved issues. In order to implement this method, we first have to retrieve a small amount of issues from Jira (configured to 50) in which the summary corresponds to the queried terms and then we apply the algorithm.

From the piece of code extracted early, we first create the corpus that consists of the concatenation of each issue summary with the description (1) and create a set of frequent words (2) in order to be able to remove them from each document (after splitting each document by white spaces - 4) . We then filter the words in order to keep only those that appear more than once (5) and build the dictionary (6). After that, we convert the corpus (each one of the documents) into a list of vectors (7).

Finally, we train the corpus with the TF-IDF algorithm (8) and perform a query with the search terms (9). It is pertinent to note that this model can be discarded or saved in order to be posteriorly trained with the help of another corpus. The last step is to return the list



```python
    def getSimilarIssues(self, issues, query):

        # (1)
        corpus = []
        for issue in issues:
            corpus.append(issue['summary'] + " " + issue['description'])

        # (2)
        stoplist = set('para a de o e para em'.split(' '))

        # (3)
        texts = [[word for word in document.lower().split() if word not in stoplist]
            for document in corpus]

        # (4)
        from collections import defaultdict
        frequency = defaultdict(int)
        for text in texts:
            for token in text:
                frequency[token] += 1

        # (5)
        processed_corpus = [[token for token in text if frequency[token] > 1] for text in texts]

        # (6)
        dictionary = corpora.Dictionary(processed_corpus)

        # (7)
        bow_corpus = [dictionary.doc2bow(text) for text in processed_corpus]

        # (8)
        tfidf = models.TfidfModel(bow_corpus)
        index = similarities.SparseMatrixSimilarity(tfidf[bow_corpus], num_features=dictionary.items.__sizeof__)


        # (9)
        query_document = query.split()
        query_bow = dictionary.doc2bow(query_document)
        sims = index[tfidf[query_bow]]

        # (10)
        return sorted(enumerate(sims), key=lambda x: x[1], reverse=True)
```

Figure 6.5: Continuous querying component algorithm

of similarities ordered by similarity. After the return the only step left is to return the top 10 similar documents.

Finally, when it comes to the FR07 in creating the release post, this only consists of a webhook that queries the Jira platform REST API to grab all the issues corresponding to the main product project and to the version specified. The issues are then parsed in order to generate the Bitrix post content and a request is made to Bitrix in order to create a new post.

6.5 Summary

The chapter explains all the steps of solution development, including both the development methodologies used and the technical aspects to be considered when programming the system. In the first section of this chapter, an overview of the implementation of the web application and the organization of the UI was discussed. As part of the second section, is presented the implementation of the backend of the system - the API, along with the algorithms used to implement continuous querying.

The following chapter will present the evaluation of the developed solution. As a result, it becomes important to show evidence that the work done resulted in an approach that is valid and has the expected quality.

Chapter 7

Experimentation & Evaluation

Once the implementation is complete, the next step is to evaluate it. This is to determine if the product developed is able to fill in the gaps identified when the problem was analyzed. It also becomes essential to check if the developed solution provides enough quality to its final users.

Several forms of evaluation should be performed when developing software. This is because the development process is incremental and takes into account the feedback given by the stakeholders when partial solutions are presented. There are, however, some evaluation parameters that can only be measured when the final (or almost final) version is developed and deployed.

With this chapter, the main objective is to evaluate the final developed solution both in terms of its technical implementation and its inherent quality as perceived by stakeholders. To achieve this, several steps must be taken into account such as the problem definition, the objectives definition, the hypothesis specification, the identification of the indicators and sources of information, and, finally, the methodology used in each of the hypotheses.

The problem and the objective were already identified in the Sections 1.2 and 1.3, respectively. The remaining items are described in this section.

7.1 Evaluation Indicators

To be able to evaluate the developed solution, the following evaluation indicators were identified:

- Functional and non-functional requirements implementation;
- Quality of the system;
- Product usability;
- Development process efficiency;
- Solution performance;

The first indicator corresponds to the full implementation of the functional and non-functional requirements. The second aims to identify the overall quality of the system as expected by the stakeholders.

The third, system usability, relates to the measurement of how the end-user can interact with the product to achieve a defined goal effectively, efficiently, and satisfactorily. Finally, the fourth indicator is intended to measure how the development process, from the perspective of the product manager, has become more efficient during the phase of analyzing the issues.

Once the evaluation indicators are determined, the hypotheses can be formulated. In the following subsections, the hypotheses formulated, that directly correlate with each of the indicators, are enunciated along with the methodology used to evaluate the respective hypothesis.

7.1.1 Hypothesis One

This hypothesis corresponds to the evaluation of whether or not the elicited requirements were fully implemented in technical terms, as described by the stakeholder.

Methodology

To evaluate this hypothesis, software tests must be implemented. The purpose of these tests is to validate that the system is operating as specified by the requirements.

To be implemented are not only unit tests to validate the proper operation of each component of the application, but also integration tests to verify the integration of the various components and systems. In the end, the main result here to consider is to have a code coverage of at least 90% to guarantee that the majority of the application is properly tested.

Results

In order to evaluate the quality of the requirements, two tools were used. For the front end project the frameworks Karma [1] and Jasmine [2] in order to perform both integration and unit tests in the Angular application. In this application, each component and service generated has a spec file associated that allows the developer to easily test each component individually.

Angular provides these testing frameworks out of the box. It enables the user to run these with a single command, making it ideal for use in a continuous integration pipeline.

Figure F.1 in Appendix F, represents the results of the tests performed in the front end application. As can be seen, this amounts to a total of 45 tests. During these tests, services were the main focus.

On the other hand, for the API the tools used to perform the tests to the various endpoints was the Postman [3] for integration tests and pytest [4] for the unit tests. Postman is a platform that provides many tools to perform tests and to generate mock servers.

In order to perform the tests in this tool, the developer must create a collection with all the HTTP requests that he wants to test in the API and then, for each request, write the assertions to be evaluated as we can see in Figure F.3, present in Appendix F.

The tests written follow an intuitive language that allows both programmers and non-programmers to seamlessly understand what is being tested. This platform becomes ideal to use when applying both Behavior Driven Development (BDD) and Test Driven Development (TDD). Table 7.1, represents the summary of the tests run along with the type and the final coverage of those tests.

[1]https://karma-runner.github.io
[2]https://jasmine.github.io
[3]https://www.postman.com
[4]https://docs.pytest.org

Table 7.1: Implemented tests summary

Project	**Test Type**	**Framework**	**Number of test cases**	**Coverage**
API (Python)	Unit	Pytest	23	95%
API (Python)	Integration	Postman	36	100%
Web App (Angular)	Unit + Integration	Karma + Jasmine	45	92%

To test the presence and correctness of the different data that should be present in each response, multiple tests were written for each endpoint of the project. In Figure F.2, in Appendix F, is presented a test run of the collection created for the project.

In the end, and by analyzing the Table 7.1, we can verify that the overall code coverage on both the projects (Web app and API) are above the established minimum of 90%. In both cases, the coverage didn't reach the maximum percentage since the tests would depend on testing basic implementation that are built in into the application. Regarding the coverage of the integration tests with Postman, these are calculated taking into account the number of API resources multiplied by the number of expected response types from each resource/endpoint.

7.1.2 Hypothesis Two

This hypothesis aims to evaluate the overall quality of the system by taking into account both the functional and non-functional requirements. In this case, the objective is that the final solution provides a quality level of 100%.

Methodology

To evaluate this hypothesis in particular, the model used corresponds to the Quantitative Evaluation Framework (QEF) model. First proposed in 2006 by Paula Escudeiro and José Bidarra [54], the QEF model represents an evaluation model that allows the improvement of the quality of the software, whilst allowing the detection and improvement of any flaws that can occur [55].

This model is based on a quality space consisting of three dimensions. Each dimension takes into account a set of factors that, in turn, group a set of requirements. Each requirement corresponds to a weight (importance), in that specific factor, which can take any even value from 0 to 10, with 10 being the highest relevance.

To evaluate the fulfillment of each requirement, a metric evaluation must be defined for each of them along with a description of its fulfillment. This must be achieved to meet that requirement. Related to each factor, there is also a correspondent relevance value that is calculated by the division of the sum of the weights of the requirements of that factor by the sum of the weights of the requirements of that dimension.

When calculating the global quality of a system, one must consider two concepts: the system ideal solution (when the system reaches its maximum quality) and the system real solution (when the system actually reaches its actual quality). By minimizing the Euclidean distance between the ideal system and the real system, we are maximizing the real quality of the solution.

For the developed project, the dimensions, factors and requirements that were defined are presented in the Table 7.2, and the application of the model is presented in the Appendix B, with the correspondent references and descriptions of each of the requirements.

Table 7.2: Dimensions, factors and requirements of the QEF model

Dimension	**Factors**	**Requirements References**
Functionality	Reporter (Actor)	FF01, FF02, FF03, FF04, FF05, FF06, FF07, FF08, FF09
	SRE Officer (Actor)	FSREO10
Supportability	Testability	ST01, ST02
	Scalability	SS01, SS02
	Maintainability	SM01, SM02, SM03, SM04, SM05
Usability	Content Quality	UQ01, UQ02, UQ03
	User Interaction	UU01, UU02, UU03, UU04, UU05, UU06
	Interface	UI01, UI02

Results

After the implementation of the solution, each one of the items mentioned in the QEF model were revisited in order to check if they were all implemented. A final model with all the values of the requirement fulfillment is shown in Appendix B. The final score for the overall solution was 100 percent.

For the functionality dimension, each one of the elicited requirements was implemented as well as the supportability dimension, with the successful implementation of integration and unit tests (as per Subsection 7.1.1), the implementation of development patterns such as builder to allow maintainability, documentation of the code and the use of linters in the front end project in order to maintain code readability. Logs in the API were also implemented to detect possible errors that might not have been detected in the tests.

As for the Usability dimension, the following was taken into consideration, among others:

- The use of appropriate language in the messages (popups and submission);
- All the forms present additional information in order to guide the user on what to input in each field;
- The interface is intuitive, this was proven with the final usability test score presented in the Subsection 7.1.3;
- All the actions that require a request to external resources (API) have a visual indicator of the processing of the action;
- Forms that allow the upload of attachments have the drag & drop functionality;

By achieving a total score of 100%, we can conclude that the implemented application exhibits the maximum degree of quality since it meets all the requirements.

7.1.3 Hypothesis Three

This hypothesis relates to the evaluation of the overall usability of the system. Usability can be described as the ease of use and acceptability of a product for a particular group of users that carry out a specific task in a specific environment [56].

The objective of this hypothesis is to determine if the overall usability of the solution is in line with the expectations of the user.

Methodology

To perform this evaluation, usability questionnaires can be applied. To conduct these surveys, a set of tasks that the user must perform must be carefully planned to focus on the objective of the usability study. Then, a questionnaire is presented to the user with a set of questions regarding the tasks at hand. These questions can be of a qualitative (open answer) and quantitative (closed answer - scale) nature.

Regarding the set of questions asked, these correspond to ten questions presented in the System Usability Scale (SUS) questionnaire, which was firstly introduced in 1984 by John Brooke to measure people's perceived usability in computer systems [57]. In terms of response scale, since they are closed questions, the Likert scale is used since they provide more reliable information than single-item scores [57]. SUS represents a single number yielding a composite measure of the overall usability of the system or product, bringing this scale from 0 to 100. Figure 7.1, shows the SUS score associated with the adjectives and acceptability scores.

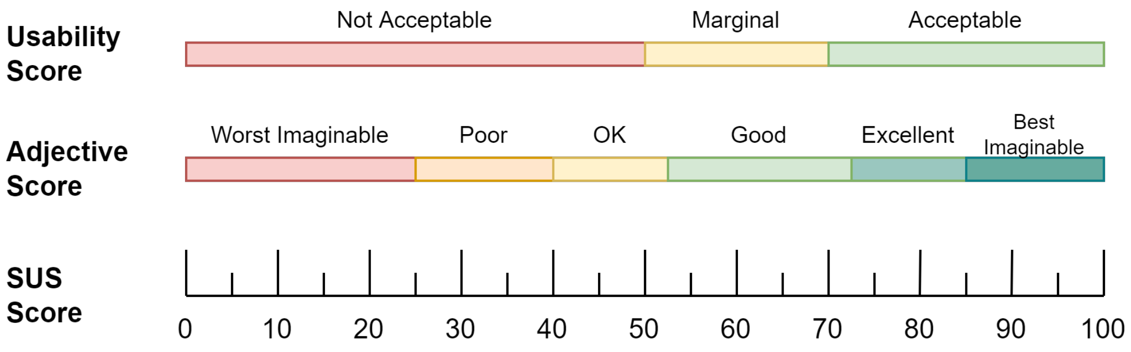


Figure 7.1: Adjectives and acceptability associated with raw SUS scores. Adapted from [58]

The model for the questionnaires conducted is presented in the Appendix C. In this questionnaire, both quantitative and qualitative answers were requested. However, the qualitative answer was optional and used mainly to gather any additional input from the interviewer. This input could be used in order to improve the platform.

Results

As stated in the methodology, the data for this hypothesis was retrieved using usability questionnaires as per Appendix C. The questionnaires were unsupervised and delivered through the Google Forms [5] service and the results can be found in the Appendix E. The total number of responses collected was 24, however four of them were discarded since the interviewers stated that they had never had contact with the platform before. This left 20 total valid answers.

In order to calculate the SUS value, the following method must be applied:

1. Convert the scale into points, by following the rule:
   - **Strongly Disagree**: 1 point
   - **Disagree**: 2 points
   - **Neutral**: 3 points
   - **Agree**: 4 points
   - **Strongly Agree**: 5 points
2. Calculate the SUS value by using the formula for each one of the responses:

   $$RawSUS = (Y + X) * 2,5$$

   ,where X = sum the points of all odd-numbered questions and subtracted by 5 and Y = 25 minus the sum of all the points of the even-numbered questions.
3. Calculate the average of all the raw SUS values for all the responses.

The first and second items are calculated in Appendix E. By calculating the third item (the average of the final SUS scores of all the responses) we end up with the **average SUS score of 83**. By comparing this value with the Figure 7.1, we can conclude that the usability score is in the **Excellent** or **Acceptable** range.

By calculating the same values for the two main departments - SAC and Product, we end up with scores of 86 and 88 respectively, which are also in the same quality range. As a result, we can conclude that the platform meets the expectations of the users by enabling them to use it easily.

Regarding the open question of suggestions and improvements, the following input was received:

- "It's much better and more intuitive";
- "Despite being able to improve from an aesthetic point of view, creating issues is simple, follows a logical path and does not present major obstacles.";

This last point was used in order to improve the visual aspects mentioned in the original answer.

7.1.4 Hypothesis Four

To evaluate whether the issue analysis performance of product managers increased during software development. This translates into the satisfaction of the users as they assess the solution's results, which, in this case, involves the diminishing of duplicate issues as well as

[5] https://docs.google.com/forms

the quality of the submitted issues. The objective here is to evaluate if the efficiency of the product managers improved when analyzing the reported issues.

Methodology

To evaluate this parameter, questionnaires can also be used, in this case, to access the overall satisfaction of the product managers with the output of the developed solution. In the same way as the usability questionnaires, the scale employed in this one is also the Likert scale. Here, both qualitative and quantitative questions were evaluated. The questionnaire used is in the Appendix D.

Results

In order to obtain data for this hypothesis, the same method as in the Subsection 7.1.3 was used, by sending the questionnaire to all the PM through the Google Forms service. A total of five responses were collected.

Since the amount of data is too small, it becomes unpractical to perform statistical analysis on them. However, since they have been reduced, one may be able to draw some conclusions directly from them. Below is a chart with the responses to each of the questions.

In Figure 7.2, we can see that, in the perception of the product managers, the ability to see similar issues when in the process of writing the summary of an issue, contributed to the reduction of similar issues written and, consequently, their posterior analysis by the PM's.

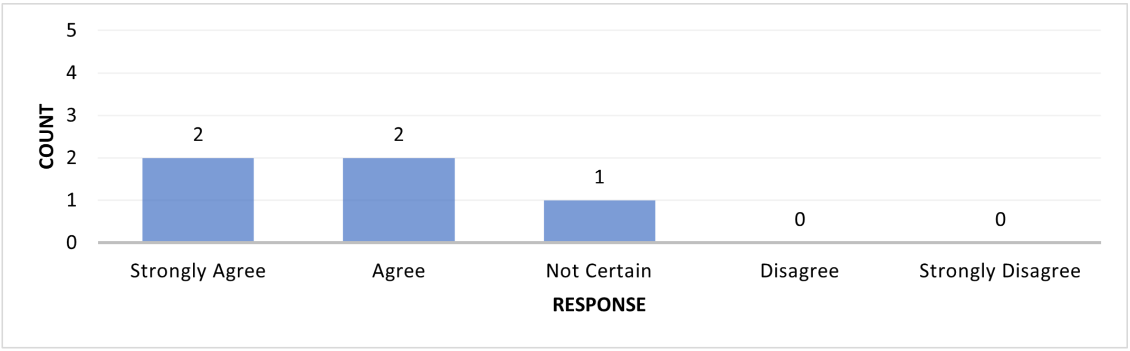


Figure 7.2: Answers for the question - Do you consider that the amount of duplicate issues has lowered over time?

In the same way, if we analyze the Figures 7.3, 7.4 and 7.5 we can conclude that both the amount of information that is included in the issue reports as well as the structure of the information has improved. This is the result of the structure enforcement imposed when writing a new issue report by the reporter.

In fact, the results from the previous Figures (7.3, 7.4 and 7.5) can be used to explain the results of the Figure 7.6, where all the product managers agree that, with the implementation of the new system the task of reviewing the issues has improved.

Regarding the open questions (comments and suggestions) from the interviewers, the following arguments were stated:

- "Its easy to use and helps in the processing of the information";
- "Could use an bit of improvement in the visual aspect (...), but the overall structure of the information is clear and straightforward"

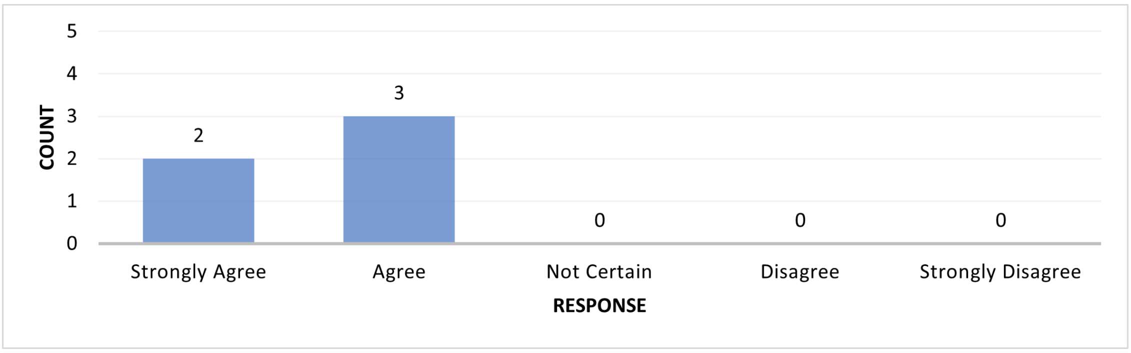


Figure 7.3: Answers for the question - Do you consider that the structure of the information improves the task of issue reviewing?

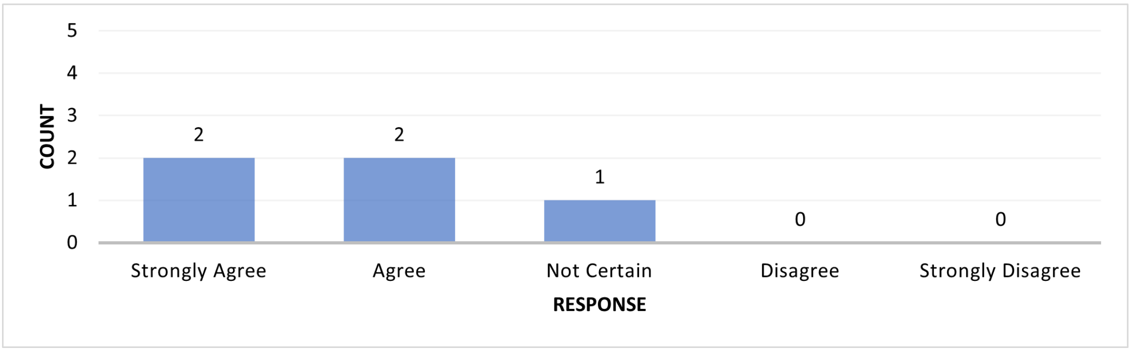


Figure 7.4: Answers for the question - Do you consider that the amount of information gathered improves the task of issue reviewing?

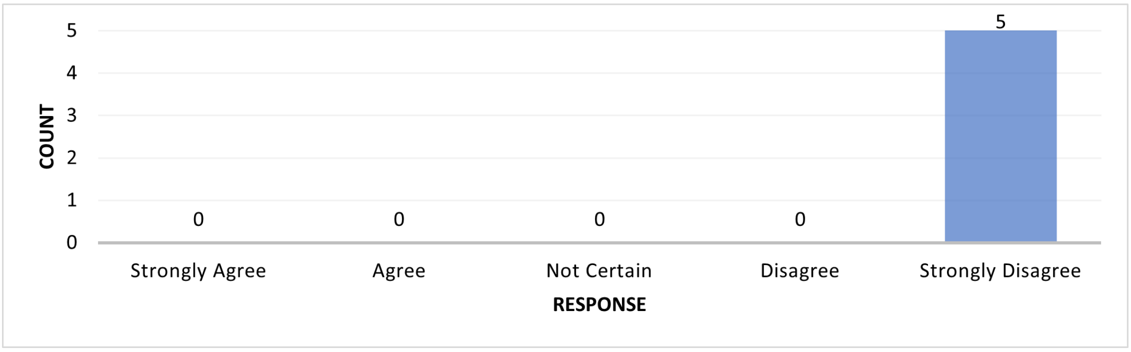


Figure 7.5: Answers for the question - Do you consider that the information in the issue description is malformed and causes confusion?

Regarding the last statement, the feedback was noted and an analysis was made in order to identify the stated problems and make the corrections in the application, similar to what was made in the previous subsection, with the feedback from the usability questionnaires.

In conclusion, upon evaluating the responses given by the product managers to the questionnaire, we can consider that, in general, the implemented solution fulfilled the objective of improving the performance of the PM's in the task of reviewing and categorizing the different issue reports.

By enforcing the need of more accurate and complete information when writing the issue reports, it helps both the PM's and developers to have all the data needed to solve the

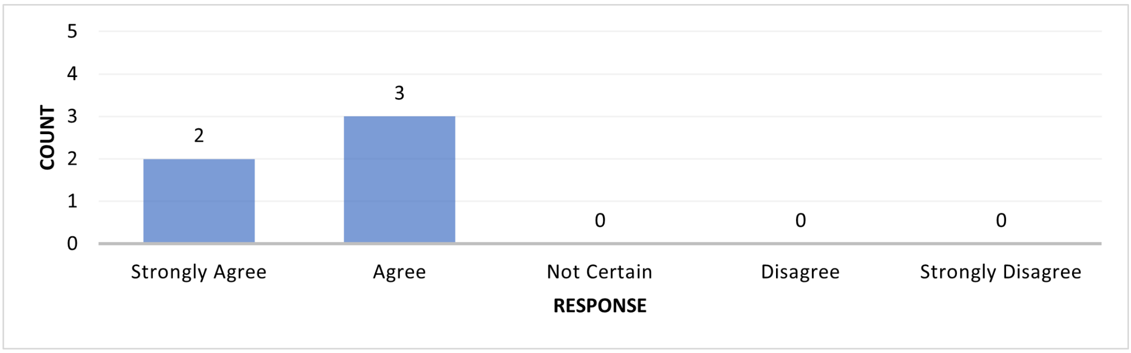


Figure 7.6: Answers for the question - Do you consider that the overall performance in issue reviewing has increased with the implementation of the new system?

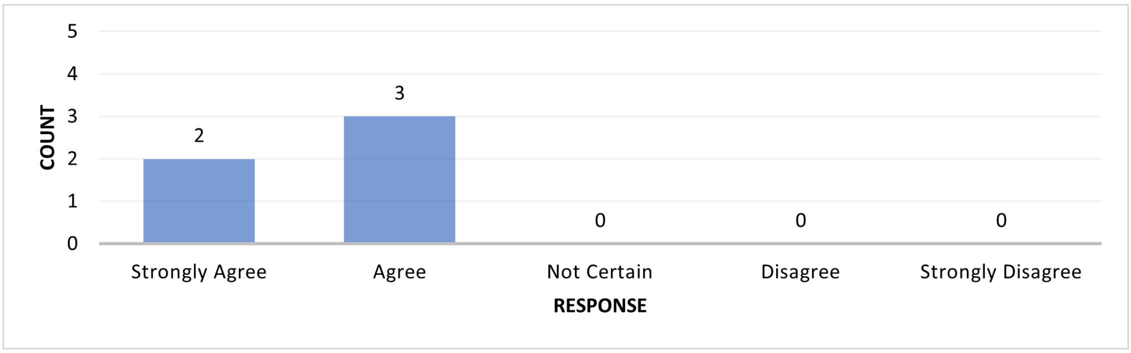


Figure 7.7: Answers for the question - Do you consider that the addition of buzz points helps in the management of the visibility of the issues?

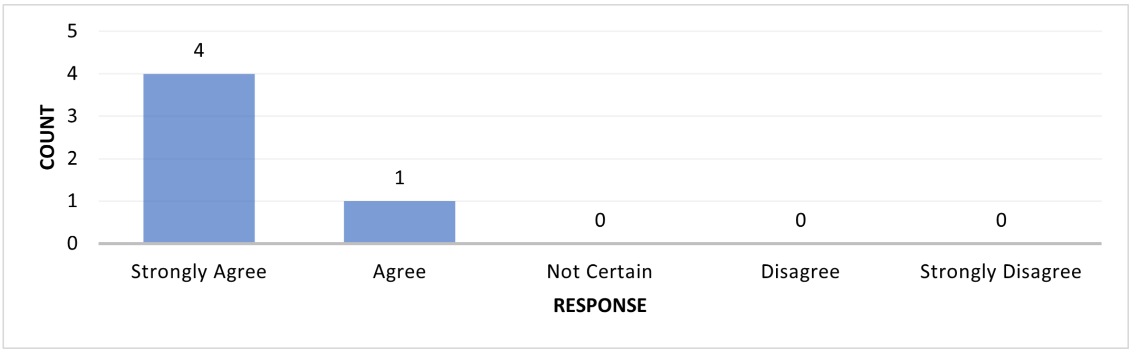


Figure 7.8: Answers for the question - Do you consider that the structure of the comments helps in the identification of the different types of comment request?

issue without the need of asking for additional clarification to the reporter. This impacts positively on the performance of the development cycle itself.

7.1.5 Hypothesis Five

In this hypothesis, the objective is to measure the performance of the API responsible for providing similar issues when using a continuous querying algorithm. It becomes pertinent to assess the amount of requests and their timeliness. This will enable one to evaluate if the chosen solution is scalable enough to support a high number of concurrent users. Because one of the company's objectives is to serve the application not only to company collaborators, but also to their clients (in a future version), this analysis is essential.

Methodology

In order to evaluate the developed solution in terms of API response times for the endpoint used to retrieve duplicate issue reports, stress tests were performed. The main objective of this test is to determine the maximum capacity of the system under test in terms of response throughput, response times and API breaking point. The main metrics here are that the API should withstand at least the number of collaborators in E-goi (150) concurrently and the response times should not be superior to 2 seconds.

In order to conduct this test, many frameworks can be used like Jmeter [6] for example, however the K6 [7] tool was used instead since it provides a straightforward implementation and has also a very comprehensive documentation base. This tool is capable of performing several parallel requests simulating several concurrent users accessing the platform per second and saving both the responses (if needed) and the statistical data that enables the user to evaluate the overall performance of the system.

Results

As defined in the previous subsection, the data needed in order to test this hypothesis was gathered using the K6 tool. The global test scenario consisted of several scenarios, each with a ramp up time of 30 seconds (the time taken for the number of concurrent virtual users to reach the maximum), a set number of virtual users (virtual threads) and a continuously stream of requests for another 30 seconds in order to simulate a continuous request stream as well. Table 7.3 contains all the six test sub-scenarios conducted and the associated number of virtual users, ramp up length and total test phase duration.

Table 7.3: Stress testing scenarios

Scenario	Virtual Users	Ramp Up Length (s)	Test Duration (s)
#1	60	30s	1m02s
#2	90	30s	1m07s
#3	120	30s	1m14s
#4	150	30s	1m18s
#5	180	30s	1m20s
#6	210	30s	1m22s

The tests performed were only directed towards testing a single endpoint and checking if it returned a "200 OK" (no evaluation of the response was made in terms of results shown). The script written enables the use of randomly generated strings in order to reproduce the search for actual terms to compare to. In terms of requests load, the script also took into consideration the gradual increase of the number of users instead of injecting a load spike.

[6] https://jmeter.apache.org
[7] https://k6.io

This simulated a more realistic traffic increase. Table 7.4 contains all the data gathered after executing the test plan.

Table 7.4: Stress testing scenarios results

Parameters/Scenario	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**
Number of Requests	799	486	553	586	617	628
Error Requests (%)	0%	0%	0%	0%	0%	2.5%
Min. Response Time (ms)	125ms	125ms	145ms	142ms	144ms	145ms
Avg. Response Time (ms)	2.43s	5.06s	10.29s	12.93s	14.9s	17.38s
90% Response Time (ms)	3.28s	14.08s	30.31s	37.79s	39.92s	51.92s
Throughput (req/ms)	6.5/s	7.2/s	7.4/s	7.5/s	7.6/s	7.6/s

From analyzing the results, we can see that the REST API is able to handle successfully the overload of having 150 concurrent users performing requests, however the response times are extremely far from ideal. This performance issue can be explained by multiple factors such as the request being dependent on the performance and availability of the Jira server, when retrieving the issues list, and the fact that the production hardware specification only count on a single-core processor with 512Mb of Random Access Memory (RAM).

For the first problem, an implementation would have to incorporate a system that could speed up the process of retrieving the list of available issue reports such as indexing them in Elastic Search. For the second problem, the developed REST API is already built taking into consideration the deployment of more instances which can then be load-balanced evenly.

7.2 Summary

In this chapter, it is evident that the development has been validated to demonstrate that it can solve the problems identified by the analysis. This validation will serve as evidence of the claim. In conducting evaluations and tests, one can assess the results of the research and draw conclusions regarding limitations and successes. These conclusions can also be used to outline future work.

Various indicators were evaluated here to validate both the quality of the development, with the implementation of software tests and the use of the QEF model. Also the benefits in the improvement of the product development cycle with the use of usability tests and the use of questionnaires proposed to the product managers were evaluated.

In the end, all the evaluations passed their projected metrics with the exception of the performance of the algorithm. This was in fact a characteristic that could have been expected with the adoption of the suggested approach. Nonetheless, this provided a platform for building a more robust solution that could be scaled if the need to expand the platform to clients arose.

Chapter 8

Conclusions

The proposal in the project includes various processes starting with the definition of objectives according to the identified problems, followed by the research of state-of-the-art technologies to create a valid solution, and finally the development and implementation of a final solution that would not only incorporate the technologies researched but also achieve each and every objective. All these steps culminated in the final planning and testing of various hypotheses that could demonstrate that the software created met the identified objectives and requirements.

8.1 Achievements

In the initial phase of the project, several objectives were identified that were in need of solving. To determine whether the development of the project resulted in a satisfactory solution, these objectives were identified. According to Section 1.3, the developed application should give a response to the following items:

1. Create a uniform structure for the information gathered in the issue reports;
2. Reduce the amount of duplicated issues;
3. Allow the distinction between issue types and provide work priority and visibility of the created issues, throughout the company;
4. Provide integration with other services present in E-goi.

Each of these items had as its main objective easing and increasing performance on the task of reviewing issues, not only by the PMs but also by developers. In order to develop this application a prior study of, not only the E-goi internal development process, Chapter 2 as well as the state of the art in terms of issue standardization, issue deduplication methods and other relevant terms (Chapter 3) were conducted.

After the acquisition of the necessary information, it was possible to perform the analysis of the problem not only in terms of the value that this solution would ultimately bring to the organization but also of the functional and non-functional requirements gathered during the elicitation phase, as per Chapter 4.

The following phase, the design phase, allowed for creating the overall structure of the application taking into account the functional requirements and envisioning alternative solutions that could be used to compare solutions and decide which is the most appropriate one to use - Chapter 5.

The implementation process took into account the best practices in terms of software development as well as in terms of patterns, project maintenance (with the development of tests), and the processes of continuous integration contained within E-goi.

Finally, after developing the solution, we were able to create distinct metrics that could allow the correct evaluation of the objectives proposed in a set of hypotheses and conclusions. With this, the following evaluation indicators were applied:

1. Correctness of the implementation of the functional and non-functional requirements;
2. Overal quality of the product;
3. Usability of the system;
4. Efficiency level in the development process;
5. Performance of the requests in conjunction with the issue detection algorithm.

For the first indicator, unit and integration tests were implemented in order to validate that both the business rules as well as the requirements elicited by the stakeholder were correctly implemented and that any change that could occur in the system could be automatically verified to check for eventual bugs. In the end all the services and functions were validated giving a satisfactory result in terms of code coverage of 92 percent. This concludes that the majority of the services and components were tested and their logic followed the specifications of the stakeholder.

In order to assess the overall quality of the product, the QEF model was used to evaluate the product on three fronts - Functionality, Supportability and Usability. The final score for all three components was 100/100, concluding that the quality gate for the application is very high.

For both the usability and performance evaluations, questionnaires were developed and delivered to the entire company (for the usability) and to all of the PM's for the performance. In the first case, a final average SUS score of 83 which is considered to be in the Acceptable or Excellent range.

Both structured (short answer questions) and unstructured (free text questions) information was collected from the PMs. Since most of the responses were within the optimal range, it turns out that the implementation provided what was proposed to solve, especially when it comes to questions regarding the structure and amount of information when submitting a new issue.

Last, but not least, the final evaluation metric was to check if the application was able to ensure that it could provide timely responses when used by the entire company. This last metric allowed us to reach the conclusion that the application can, in fact, be used by the entire company (in terms of concurrent requests) however performance is a major issue.

Even though this could have a multitude of factors from hardware limitations, in terms of CPU and memory usage, (that could be surpassed by increasing the server capabilities) to implementation issues since the API is dependent on the performance of the Jira service to retrieve the multitude of issues present (in which another solution would be to index all the issues in Elastic Search to increase the performance of document retrieval).

This last one has proven to be a major limitation of the application, however, if the provided solutions do not improve the performance, the alternative solution envisioned in Section

5.4 could be applied, where the evaluation of the duplicate nature of the reported issues is performed after they have been submitted.

AS a final remark, and since the majority of the evaluate metrics passed their test, one can conclude that the implemented solution does in fact provide a solution to the analyzed problem and takes into consideration all the requirements proposed by the stakeholder.

8.2 Future Work

To date, the application is in use by the company in all departments. This has led to more than 300 issues being created with the platform in a relatively short time span. It became visible that the issue resolution had an overall improvement specially in the analysis of the information present in the issues reports by the product managers. In fact, when questioned about the utility of the platform to a product manager, the following input was received:

> "The issue creator helps us in the creation and analysis of the issues in a global way, because the ones that create new issues have a better perception of the quantity and quality of the information that is delivered and those who analyze it receive the information already organized"

The overall success of the project has been achieved, but there is still room for improvement, and the E-goi clients have to play a more active role in the detection and resolution of problems and improvements.

As part of our discussions with the CTO, one of the main goals in the future is to enable E-goi clients to submit and track their issues. Naturally, this would require an additional persistence layer and communication with other internal services. It would also lead to an exponential creation of duplicates, which would have to be taken into consideration and implement a more robust solution to the detection of such (the alternative solution present in the design chapter, Chapter 5, could be a suitable candidate).

Lastly, other improvements that were also in the making and are planned for the future, is the creation of new report types such as incidents and tasks among others. This could allow a more precise control over the finality of each one of them.

Bibliography

[1] Jiehui Ju et al. "Research on key technology in SaaS". In: *Proceedings - 2010 International Conference on Intelligent Computing and Cognitive Informatics, ICICCI 2010*. 2010, pp. 384–387. isbn: 9780769540146. doi: 10.1109/ICICCI.2010.120.

[2] IIBA. *Guide to the Business Analysis Body of Knowledge (BABOK)*. 2015, p. 514. isbn: 9781927584033.

[3] Gurpreet Singh Matharu et al. "Empirical Study of Agile Software Development Methodologies". In: *ACM SIGSOFT Software Engineering Notes* 40.1 (Feb. 2015), pp. 1–6. issn: 0163-5948. doi: 10.1145/2693208.2693233.

[4] Atlassian. *What are issue types ?* 2020. url: https://support.atlassian.com/jira-cloud-administration/docs/what-are-issue-types/ (visited on 01/31/2022).

[5] Tommaso Dal Sasso, Andrea Mocci, and Michele Lanza. "What Makes a Satisficing Bug Report?" In: *Proceedings - 2016 IEEE International Conference on Software Quality, Reliability and Security, QRS 2016*. 2016, pp. 164–174. isbn: 9781509041275. doi: 10.1109/QRS.2016.28.

[6] Thomas Zimmermann et al. "What makes a good bug report?" In: *IEEE Transactions on Software Engineering* 36.5 (2010), pp. 618–643. issn: 00985589. doi: 10.1109/TSE.2010.63.

[7] Mozhan Soltani, Felienne Hermans, and Thomas Bäck. "The significance of bug report elements". In: *Empirical Software Engineering* 25.6 (2020), pp. 5255–5294. issn: 15737616. doi: 10.1007/s10664-020-09882-z.

[8] Steven Davies and Marc Roper. "What's in a bug report?" In: *International Symposium on Empirical Software Engineering and Measurement*. 2014. isbn: 9781450327749. doi: 10.1145/2652524.2652541.

[9] Md Rejaul Karim. "Key features recommendation to improve bug reporting". In: *Proceedings - 2019 IEEE/ACM International Conference on Software and System Processes, ICSSP 2019*. Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 1–4. isbn: 9781728133935. doi: 10.1109/ICSSP.2019.00010.

[10] Md Rejaul Karim et al. "Identifying and predicting key features to support bug reporting". In: *Journal of Software: Evolution and Process* 31.12 (Dec. 2019). issn: 20477481. doi: 10.1002/smr.2184.

[11] Petra Heck and Andy Zaidman. "A framework for quality assessment of just-in-time requirements: the case of open source feature requests". In: *Requirements Engineering* 22.4 (Nov. 2017), pp. 453–473. issn: 1432010X. doi: 10.1007/s00766-016-0247-5.

[12] Yguaratã Cerqueira Cavalcanti et al. "The bug report duplication problem: An exploratory study". In: *Software Quality Journal* 21.1 (2013), pp. 39–66. issn: 15731367. doi: 10.1007/s11219-011-9164-5.

[13] Som Gupta and Sanjai Kumar Gupta. "A Systematic Study of Duplicate Bug Report Detection". In: *International Journal of Advanced Computer Science and Applications* 12.1 (2021), pp. 578–589. issn: 21565570. doi: `10.14569/IJACSA.2021.0120167`.

[14] Yguaratã Cerqueira Cavalcanti et al. "One step more to understand the bug report duplication problem". In: *Proceedings - 24th Brazilian Symposium on Software Engineering, SBES 2010*. 2010, pp. 148–157. isbn: 9780769542737. doi: `10.1109/SBES.2010.12`.

[15] Nicolas Bettenburg et al. "Duplicate bug reports considered harmful... Really?" In: *IEEE International Conference on Software Maintenance, ICSM*. 2008, pp. 337–345. isbn: 9781424426140. doi: `10.1109/ICSM.2008.4658082`.

[16] Abram Hindle and Curtis Onuczko. "Preventing duplicate bug reports by continuously querying bug reports". In: *Empirical Software Engineering* 24.2 (2019), pp. 902–936. issn: 15737616. doi: `10.1007/s10664-018-9643-4`.

[17] Zahra Aminoroaya, Behzad Soleimani Neysiani, and Mohammad Hossein Nadimi Shahraki. "Detecting Duplicate Bug Reports Techniques". In: *Research Journal of Applied Sciences* 13.9 (2019), pp. 522–531. issn: 1815932X. doi: `10.36478/rjasci.2018.522.531`.

[18] Olga Baysal, Reid Holmes, and Michael W Godfrey. "Situational awareness: Personalizing issue tracking systems". In: *Proceedings - International Conference on Software Engineering*. 2013, pp. 1185–1188. isbn: 9781467330763. doi: `10.1109/ICSE.2013.6606674`.

[19] Davide Falessi, Freddy Hernandez, and Foaad Khosmood. "Issue tracking systems: What developers want and use". In: *ICSOFT 2018 - Proceedings of the 13th International Conference on Software Technologies*. 2019, pp. 543–548. isbn: 9789897583209. doi: `10.5220/0006818405430548`.

[20] Dane Bertram et al. "Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams". In: *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*. 2010, pp. 291–300. isbn: 9781605589879. doi: `10.1145/1718918.1718972`.

[21] *Jira*. url: `https://www.atlassian.com/software/jira` (visited on 02/01/2022).

[22] *Redmine*. url: `https://www.redmine.org/` (visited on 02/01/2022).

[23] *Bugzilla*. url: `https://www.bugzilla.org/` (visited on 02/01/2022).

[24] Mohamed Sami Rakha et al. "Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval". In: 23 (2018), pp. 2597–2621. doi: `10.1007/s10664-017-9590-5`.

[25] Aniruddha Prakash Kshirsagar and Pankaj R Chandre. "Issue Tracking System with Duplicate Issue Detection". In: (2015). doi: `10.1145/2818567.2818575`.

[26] Fangwen Mu et al. "NERO: A Text-based Tool for Content Annotation and Detection of Smells in Feature Requests". In: *Proceedings of the IEEE International Conference on Requirements Engineering* 2020-Augus (2020), pp. 400–403. issn: 23326441. doi: `10.1109/RE48521.2020.00056`.

[27] Iosif Alvertis et al. “User Involvement in Software Development Processes”. In: *Procedia Computer Science*. Vol. 97. 2016, pp. 73–83. doi: `10.1016/j.procs.2016.08.282`.

[28] Pilar Carbonell, Ana I. Rodríguez-Escudero, and Devashish Pujari. “Customer involvement in new service development: An examination of antecedents and outcomes”. In: *Journal of Product Innovation Management* 26.5 (2009), pp. 536–550. issn: 07376782. doi: `10.1111/j.1540-5885.2009.00679.x`.

[29] Rashina Hoda, James Noble, and Stuart Marshall. “The impact of inadequate customer collaboration on self-organizing Agile teams”. In: *Information and Software Technology*. Vol. 53. 5. Elsevier, May 2011, pp. 521–534. doi: `10.1016/j.infsof.2010.10.009`.

[30] Michele Chinosi and Alberto Trombetta. “BPMN: An introduction to the standard”. In: *Computer Standards and Interfaces* 34.1 (2012), pp. 124–134. issn: 09205489. doi: `10.1016/j.csi.2011.06.002`.

[31] Gustav Aagesen and John Krogstie. “Analysis and Design of Business Processes Using BPMN”. In: *Handbook on Business Process Management 1*. Springer Berlin Heidelberg, 2010, pp. 213–235. doi: `10.1007/978-3-642-00416-2\_10`.

[32] Lawrences D. Miles. *Technique of Value Analysis and Engineering*. Jan. 1989.

[33] Marjan Leber et al. “Value analysis as an integral part of new product development”. In: *Procedia Engineering*. Vol. 69. 2014, pp. 90–98. doi: `10.1016/j.proeng.2014.02.207`.

[34] Hamid Tohidi and Mohammad Mehdi Jabbari. “The important of Innovation and its Crucial Role in Growth, Survival and Success of Organizations”. In: *Procedia Technology* 1 (2012), pp. 535–538. issn: 22120173. doi: `10.1016/j.protcy.2012.02.116`.

[35] Buraj Patrakosol and David L Olson. “How interfirm collaboration benefits IT innovation”. In: *Information and Management* 44.1 (2007), pp. 53–62. issn: 03787206. doi: `10.1016/j.im.2006.10.003`.

[36] PA Koen et al. *Fuzzy front end: Effective methods, tools, and techniques*. 2002.

[37] Cornelius Herstatt and Birgit Verworn. “The ‘Fuzzy Front End’ of Innovation”. In: *Bringing Technology and Innovation into the Boardroom*. Palgrave Macmillan UK, 2004, pp. 347–372. doi: `10.1057/9780230512771_16`.

[38] Peter Koen et al. “Providing clarity and a common language to the "fuzzy front end"”. In: *Research Technology Management* 44.2 (2001), pp. 46–55. issn: 08956308. doi: `10.1080/08956308.2001.11671418`.

[39] Abdelrahman E. M. Ezzat and Hesham S Hamoud. “Analytic hierarchy process as module for productivity evaluation and decision-making of the operation theater”. In: *Avicenna Journal of Medicine* 06.01 (2016), pp. 3–7. issn: 2231-0770. doi: `10.4103/2231-0770.173579`.

[40] Omkarprasad S. Vaidya and Sushil Kumar. “Analytic hierarchy process: An overview of applications”. In: *European Journal of Operational Research* 169.1 (2006), pp. 1–29. issn: 03772217. doi: `10.1016/j.ejor.2004.04.028`.

[41] David P. Lepak, Ken G. Smith, and M. Susan Taylor. *Value creation and value capture: A multilevel perspective*. 2007. doi: 10.5465/AMR.2007.23464011.

[42] Constantinos Coursaris, Khaled Hassanein, and Milena Head. "Mobile technologies and the value chain: Participants, activities and value creation". In: *International Conference on Mobile Business, ICMB 2006*. 2006, p. 8. isbn: 0769525954. doi: 10.1109/ICMB.2006.35.

[43] Almoatazbillah Hassan. "The Value Proposition Concept in Marketing: How Customers Perceive the Value Delivered by Firms– A Study of Customer Perspectives on Supermarkets in Southampton in the United Kingdom". In: *International Journal of Marketing Studies* 4.3 (2012). issn: 1918-719X. doi: 10.5539/ijms.v4n3p68.

[44] Gillian Pritchett. "Value Proposition Design: How to Create Products and Services Customers Want". In: *Central European Business Review* 3.4 (2014), pp. 52–52. issn: 18054854. doi: 10.18267/j.cebr.104.

[45] Lachana Ramingwong. "A review of requirements engineering processes, problems and models." In: *International Journal of Engineering Science and Technology* 4 (6 2012).

[46] Saad Alsaleh and Haryani Haron. "The Most Important Functional and Non-Functional Requirements of Knowledge Sharing System at Public Academic Institutions: A Case Study". In: *Lecture Notes on Software Engineering* 4 (2 2016). issn: 23013559. doi: 10.7763/lnse.2016.v4.242.

[47] Sulaiman Aljarallah and Russell Lock. "A Comparison of Software Quality Characteristics and Software Sustainability Characteristics". In: *ACM International Conference Proceeding Series* (Sept. 2019). doi: 10.1145/3386164.3389078.

[48] Ian Sommerville. *Software engineering / Ian Sommerville. – 9th ed.* Vol. 9. 2011.

[49] *ElasticSearch*. url: https://www.elastic.co/what-is/elasticsearch.

[50] *Angular*. url: https://angular.io/guide/what-is-angular/.

[51] *The Falcon Web Framework*. url: https://falcon.readthedocs.io/en/stable/.

[52] *Gensim*. url: https://radimrehurek.com/gensim/.

[53] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". In: *IEEE Access* 5 (2017). issn: 21693536. doi: 10.1109/ACCESS.2017.2685629.

[54] Paula Escudeiro and José Bidarra. "X-TEC: Techno-didactical extension for instruction/learning based on computer: A new development model for Educational Software". In: *WEBIST 2006 - 2nd International Conference on Web Information Systems and Technologies, Proceedings*. Vol. SEBEG. EL/-. 2006, pp. 325–331. isbn: 9789728865474. doi: 10.5220/0001247503250331.

[55] Paula Escudeiro and José Bidarra. "Quantitative Evaluation Framework (QEF)". In: *Revista Ibérica de Sistemas e Tecnologias de Informação* Volume 1 (2008).

[56] James R Lewis. "Usability Testing". In: (2006).

[57] James R. Lewis. "The System Usability Scale: Past, Present, and Future". In: *International Journal of Human-Computer Interaction* 34 (7 2018). issn: 15327590. doi: 10.1080/10447318.2018.1455307.

[58] Aaron Bangor, Philip T. Kortum, and James T. Miller. "An empirical evaluation of the system usability scale". In: *International Journal of Human-Computer Interaction* 24 (6 Aug. 2008), pp. 574–594. issn: 10447318. doi: 10.1080/10447310802205776.

Appendix A

Software Development Process Sub-Processes

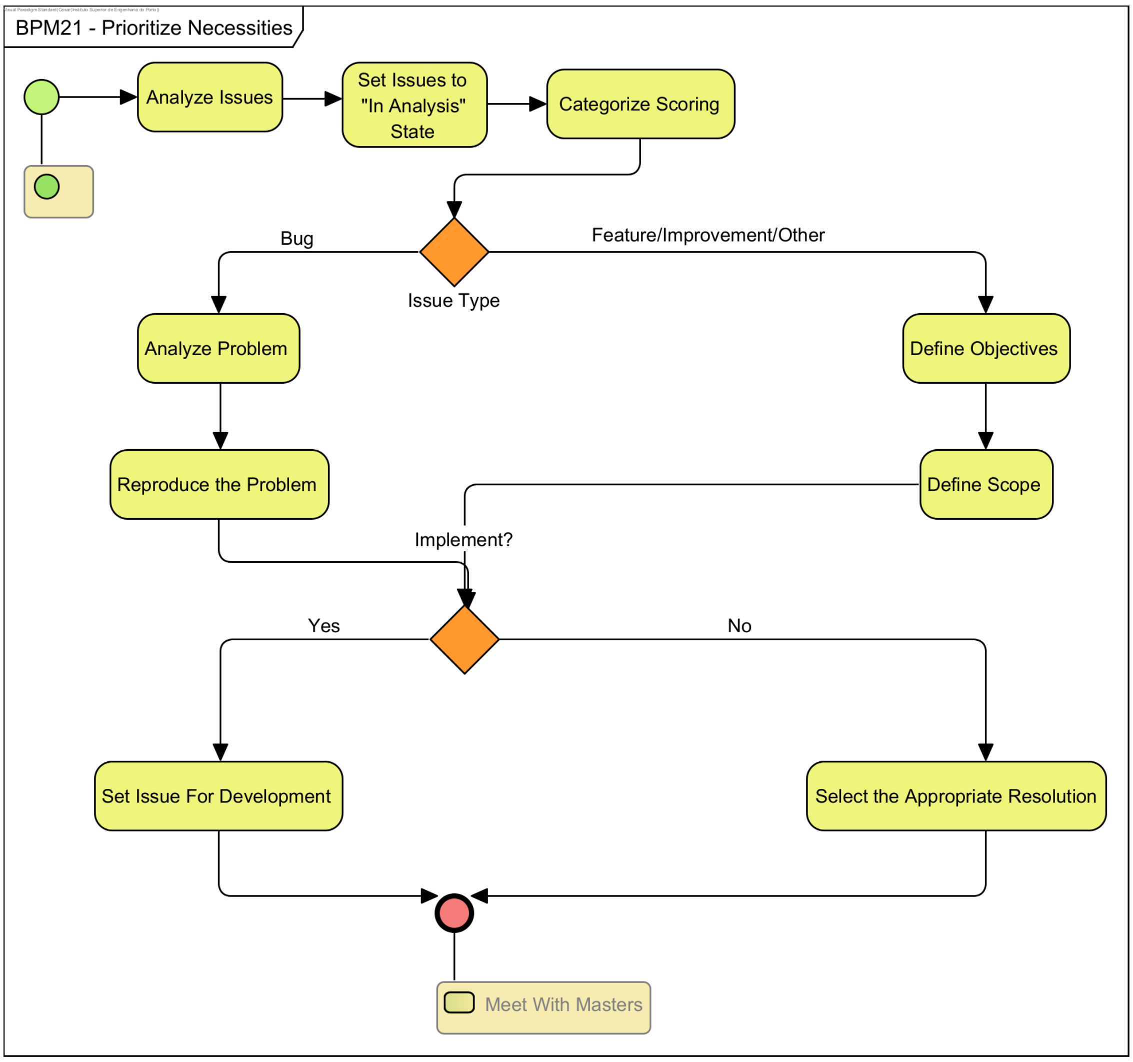

BPM21 - Prioritize Necessities
Analyze Issues
Set Issues to "In Analysis" State
Categorize Scoring
Bug
Feature/Improvement/Other
Issue Type
Analyze Problem
Define Objectives
Reproduce the Problem
Define Scope
Implement?
Yes
No
Set Issue For Development
Select the Appropriate Resolution
Meet With Masters

Figure A.1: Necessities prioritization sub-process

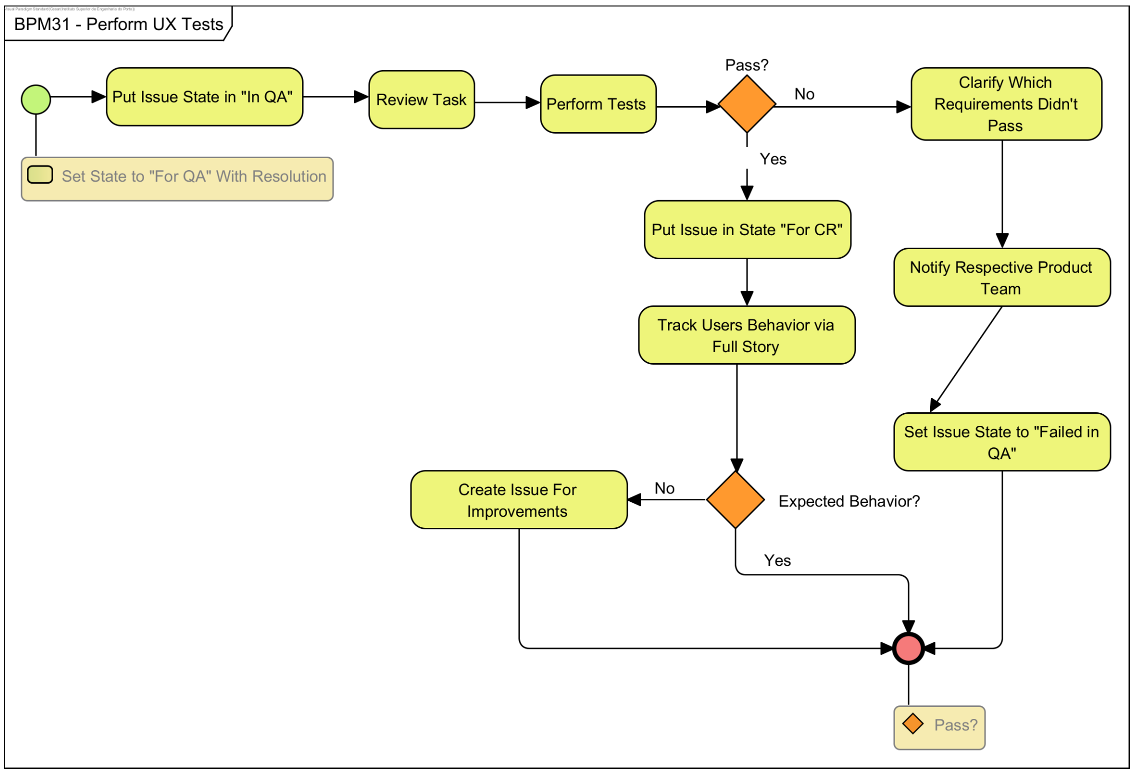

BPM31 - Perform UX Tests
Put Issue State in "In QA"
Review Task
Perform Tests
Pass?
No
Clarify Which Requirements Didn't Pass
Yes
Put Issue in State "For CR"
Notify Respective Product Team
Track Users Behavior via Full Story
Set Issue State to "Failed in QA"
Create Issue For Improvements
No
Expected Behavior?
Yes
Set State to "For QA" With Resolution
Pass?

Figure A.2: Perform usability tests sub-process

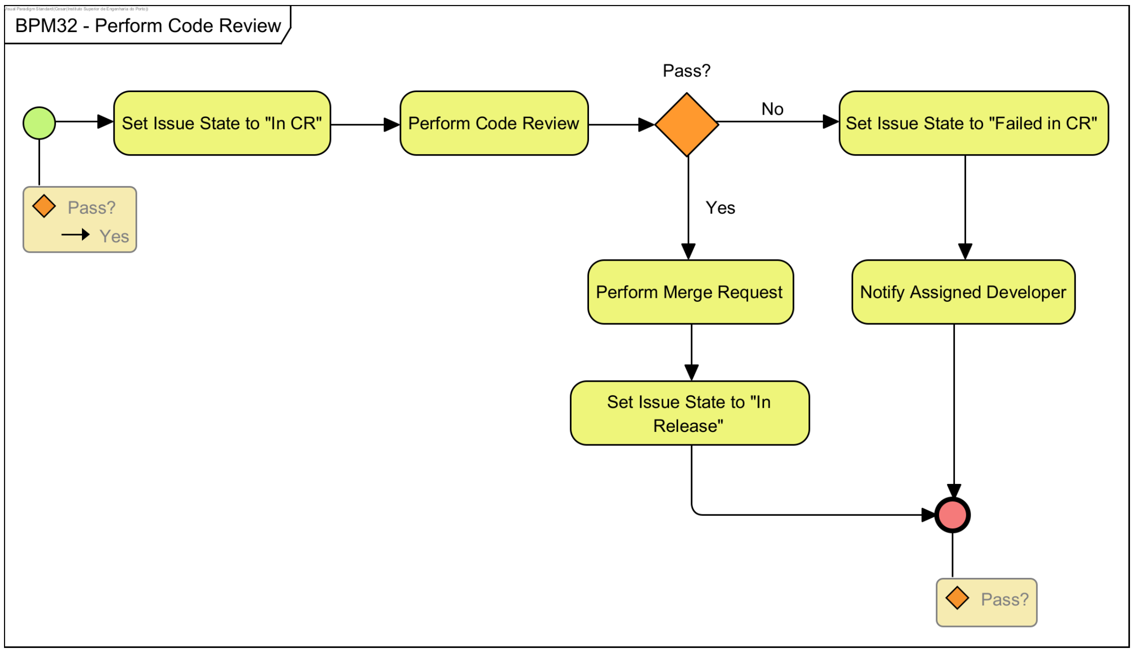

BPM32 - Perform Code Review
Set Issue State to "In CR"
Perform Code Review
Pass?
No
Set Issue State to "Failed in CR"
Yes
Pass?
Yes
Perform Merge Request
Notify Assigned Developer
Set Issue State to "In Release"
Pass?

Figure A.3: Perform code review sub-process

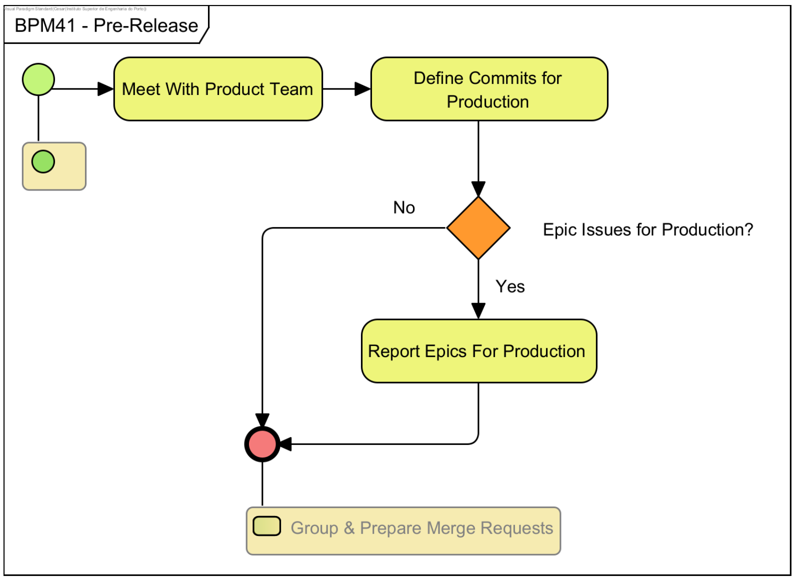

BPM41 - Pre-Release
Meet With Product Team
Define Commits for Production
No
Epic Issues for Production?
Yes
Report Epics For Production
Group & Prepare Merge Requests

Figure A.4: Pre-release sub-process

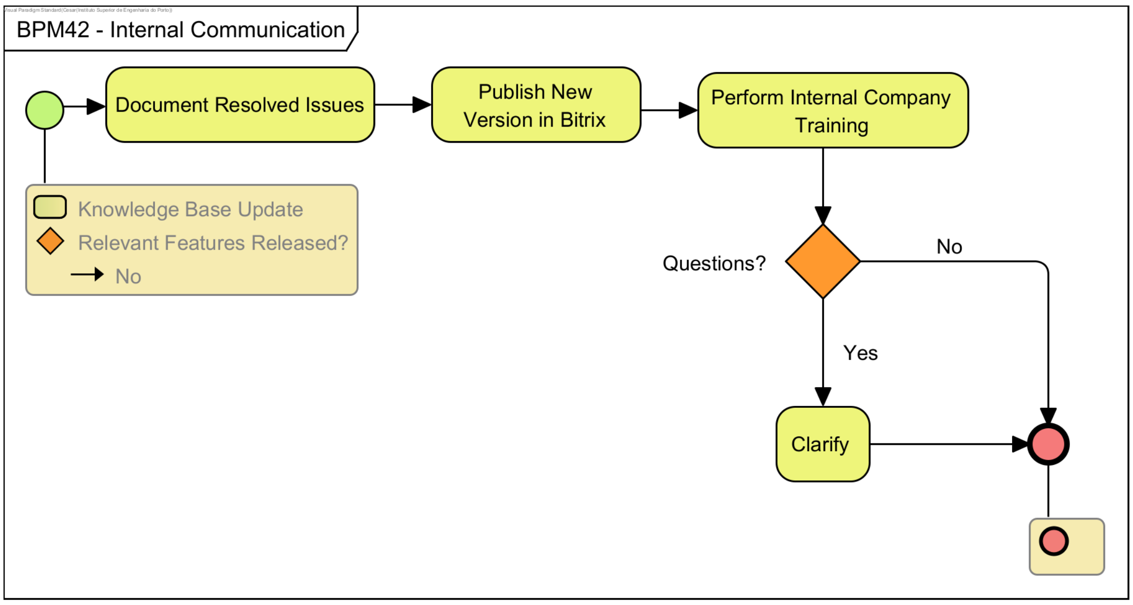
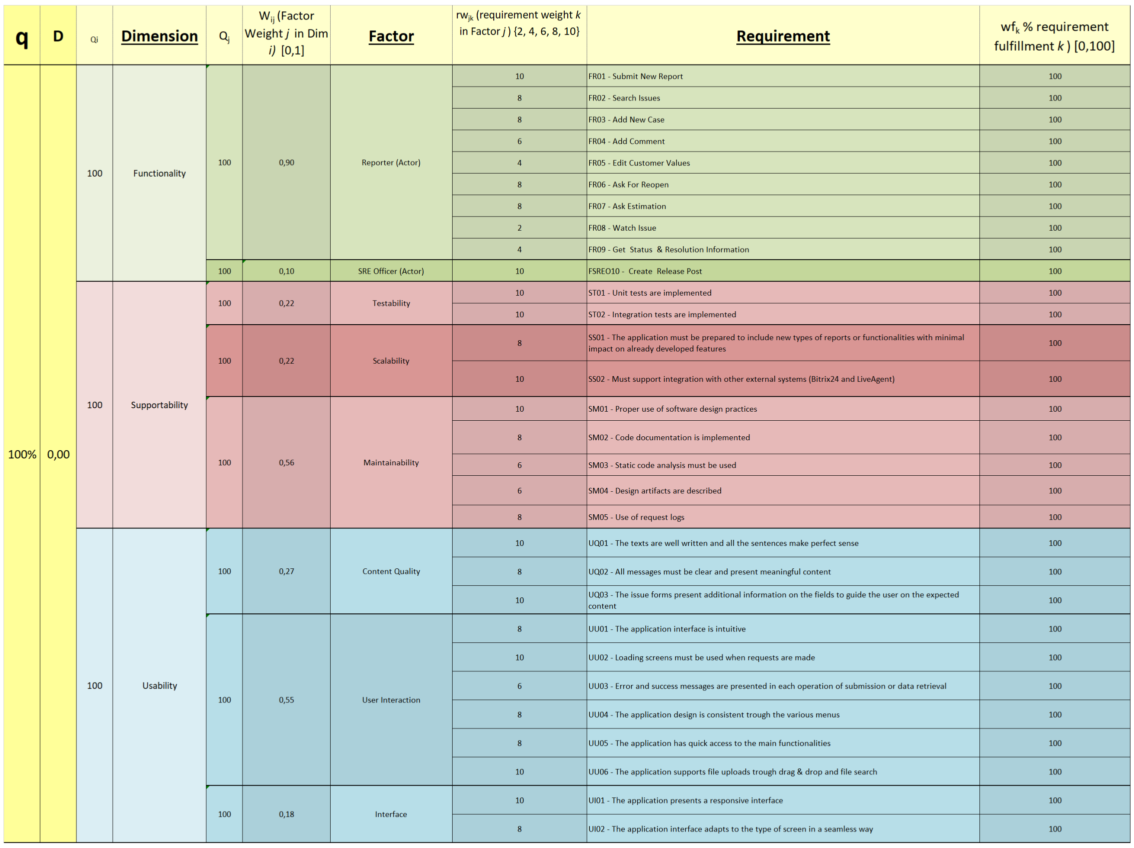

BPM42 - Internal Communication
Document Resolved Issues
Publish New Version in Bitrix
Perform Internal Company Training
Knowledge Base Update
Relevant Features Released?
No
Questions?
No
Yes
Clarify

Figure A.5: Internal communication sub-process

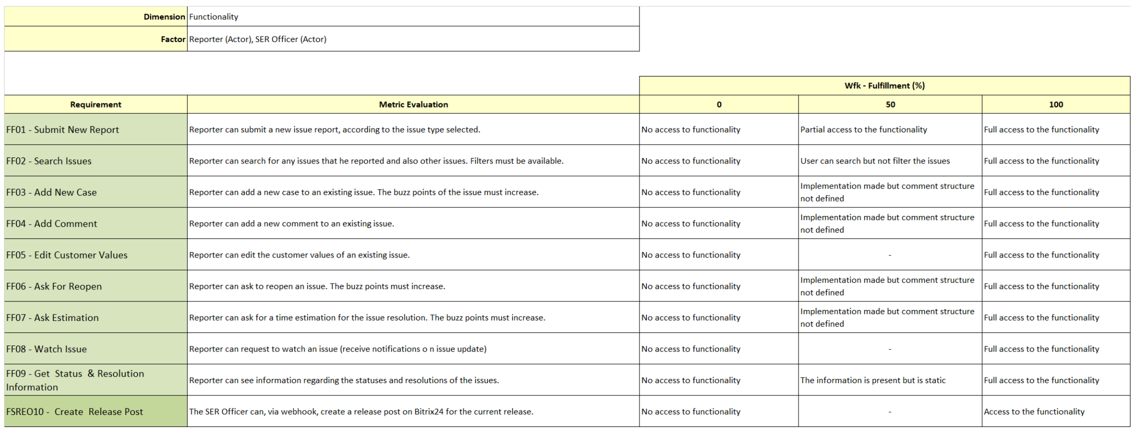
Appendix B

QEF Model



q	D	Qi	Dimension	Q_j	W_{ij} (Factor Weight *j* in Dim *i*) [0,1]	Factor	rw_{jk} (requirement weight *k* in Factor *j*) {2, 4, 6, 8, 10}	Requirement	wf_k % requirement fulfillment *k*) [0,100]
100%	0,00	100	Functionality	100	0,90	Reporter (Actor)	10	FR01 - Submit New Report	100
							8	FR02 - Search Issues	100
							8	FR03 - Add New Case	100
							6	FR04 - Add Comment	100
							4	FR05 - Edit Customer Values	100
							8	FR06 - Ask For Reopen	100
							8	FR07 - Ask Estimation	100
							2	FR08 - Watch Issue	100
							4	FR09 - Get Status & Resolution Information	100
				100	0,10	SRE Officer (Actor)	10	FSREO10 - Create Release Post	100
		100	Supportability	100	0,22	Testability	10	ST01 - Unit tests are implemented	100
							10	ST02 - Integration tests are implemented	100
				100	0,22	Scalability	8	SS01 - The application must be prepared to include new types of reports or functionalities with minimal impact on already developed features	100
							10	SS02 - Must support integration with other external systems (Bitrix24 and LiveAgent)	100
				100	0,56	Maintainability	10	SM01 - Proper use of software design practices	100
							8	SM02 - Code documentation is implemented	100
							6	SM03 - Static code analysis must be used	100
							6	SM04 - Design artifacts are described	100
							8	SM05 - Use of request logs	100
		100	Usability	100	0,27	Content Quality	10	UQ01 - The texts are well written and all the sentences make perfect sense	100
							8	UQ02 - All messages must be clear and present meaningful content	100
							10	UQ03 - The issue forms present additional information on the fields to guide the user on the expected content	100
				100	0,55	User Interaction	8	UU01 - The application interface is intuitive	100
							10	UU02 - Loading screens must be used when requests are made	100
							6	UU03 - Error and success messages are presented in each operation of submission or data retrieval	100
							8	UU04 - The application design is consistent trough the various menus	100
							8	UU05 - The application has quick access to the main functionalities	100
							10	UU06 - The application supports file uploads trough drag & drop and file search	100
				100	0,18	Interface	10	UI01 - The application presents a responsive interface	100
							8	UI02 - The application interface adapts to the type of screen in a seamless way	100

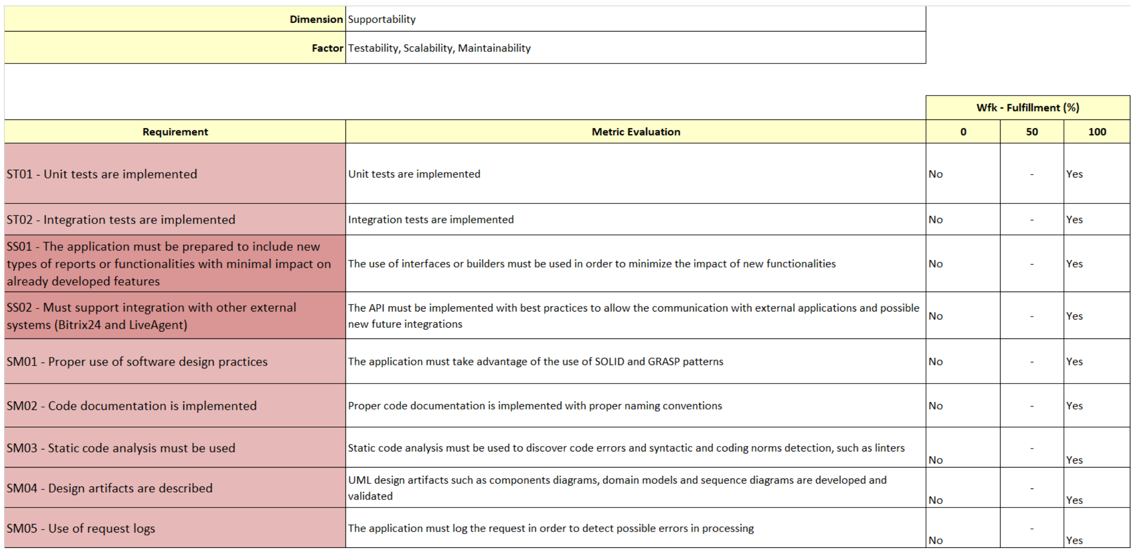
Figure B.1: QEF overall picture with dimensions, factors and requirements



Dimension	Functionality
Factor	Reporter (Actor), SER Officer (Actor)

		Wfk - Fulfillment (%)		
Requirement	**Metric Evaluation**	**0**	**50**	**100**
FF01 - Submit New Report	Reporter can submit a new issue report, according to the issue type selected.	No access to functionality	Partial access to the functionality	Full access to the functionality
FF02 - Search Issues	Reporter can search for any issues that he reported and also other issues. Filters must be available.	No access to functionality	User can search but not filter the issues	Full access to the functionality
FF03 - Add New Case	Reporter can add a new case to an existing issue. The buzz points of the issue must increase.	No access to functionality	Implementation made but comment structure not defined	Full access to the functionality
FF04 - Add Comment	Reporter can add a new comment to an existing issue.	No access to functionality	Implementation made but comment structure not defined	Full access to the functionality
FF05 - Edit Customer Values	Reporter can edit the customer values of an existing issue.	No access to functionality	-	Full access to the functionality
FF06 - Ask For Reopen	Reporter can ask to reopen an issue. The buzz points must increase.	No access to functionality	Implementation made but comment structure not defined	Full access to the functionality
FF07 - Ask Estimation	Reporter can ask for a time estimation for the issue resolution. The buzz points must increase.	No access to functionality	Implementation made but comment structure not defined	Full access to the functionality
FF08 - Watch Issue	Reporter can request to watch an issue (receive notifications o n issue update)	No access to functionality	-	Full access to the functionality
FF09 - Get Status & Resolution Information	Reporter can see information regarding the statuses and resolutions of the issues.	No access to functionality	The information is present but is static	Full access to the functionality
FSREO10 - Create Release Post	The SER Officer can, via webhook, create a release post on Bitrix24 for the current release.	No access to functionality	-	Access to the functionality

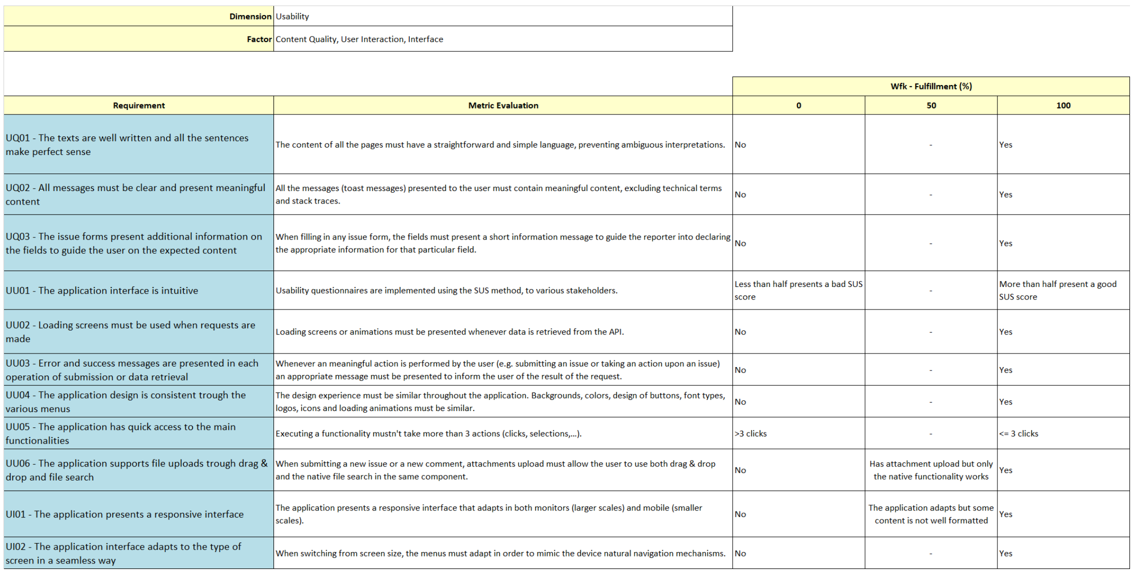
Figure B.2: Evaluation Indicators for the factors of the functionality dimension



Dimension	Supportability
Factor	Testability, Scalability, Maintainability

		Wfk - Fulfillment (%)		
Requirement	**Metric Evaluation**	**0**	**50**	**100**
ST01 - Unit tests are implemented	Unit tests are implemented	No	-	Yes
ST02 - Integration tests are implemented	Integration tests are implemented	No	-	Yes
SS01 - The application must be prepared to include new types of reports or functionalities with minimal impact on already developed features	The use of interfaces or builders must be used in order to minimize the impact of new functionalities	No	-	Yes
SS02 - Must support integration with other external systems (Bitrix24 and LiveAgent)	The API must be implemented with best practices to allow the communication with external applications and possible new future integrations	No	-	Yes
SM01 - Proper use of software design practices	The application must take advantage of the use of SOLID and GRASP patterns	No	-	Yes
SM02 - Code documentation is implemented	Proper code documentation is implemented with proper naming conventions	No	-	Yes
SM03 - Static code analysis must be used	Static code analysis must be used to discover code errors and syntactic and coding norms detection, such as linters	No	-	Yes
SM04 - Design artifacts are described	UML design artifacts such as components diagrams, domain models and sequence diagrams are developed and validated	No	-	Yes
SM05 - Use of request logs	The application must log the request in order to detect possible errors in processing	No	-	Yes

Figure B.3: Evaluation Indicators for the factors of the supportability dimension



Dimension	Usability
Factor	Content Quality, User Interaction, Interface

		Wfk - Fulfillment (%)		
Requirement	**Metric Evaluation**	**0**	**50**	**100**
UQ01 - The texts are well written and all the sentences make perfect sense	The content of all the pages must have a straightforward and simple language, preventing ambiguous interpretations.	No	-	Yes
UQ02 - All messages must be clear and present meaningful content	All the messages (toast messages) presented to the user must contain meaningful content, excluding technical terms and stack traces.	No	-	Yes
UQ03 - The issue forms present additional information on the fields to guide the user on the expected content	When filling in any issue form, the fields must present a short information message to guide the reporter into declaring the appropriate information for that particular field.	No	-	Yes
UU01 - The application interface is intuitive	Usability questionnaires are implemented using the SUS method, to various stakeholders.	Less than half presents a bad SUS score	-	More than half present a good SUS score
UU02 - Loading screens must be used when requests are made	Loading screens or animations must be presented whenever data is retrieved from the API.	No	-	Yes
UU03 - Error and success messages are presented in each operation of submission or data retrieval	Whenever an meaningful action is performed by the user (e.g. submitting an issue or taking an action upon an issue) an appropriate message must be presented to inform the user of the result of the request.	No	-	Yes
UU04 - The application design is consistent trough the various menus	The design experience must be similar throughout the application. Backgrounds, colors, design of buttons, font types, logos, icons and loading animations must be similar.	No	-	Yes
UU05 - The application has quick access to the main functionalities	Executing a functionality mustn't take more than 3 actions (clicks, selections,...).	>3 clicks	-	<= 3 clicks
UU06 - The application supports file uploads trough drag & drop and file search	When submitting a new issue or a new comment, attachments upload must allow the user to use both drag & drop and the native file search in the same component.	No	Has attachment upload but only the native functionality works	Yes
UI01 - The application presents a responsive interface	The application presents a responsive interface that adapts in both monitors (larger scales) and mobile (smaller scales).	No	The application adapts but some content is not well formatted	Yes
UI02 - The application interface adapts to the type of screen in a seamless way	When switching from screen size, the menus must adapt in order to mimic the device natural navigation mechanisms.	No	-	Yes

Figure B.4: Evaluation Indicators for the factors of the usability dimension

Appendix C

Usability Questionnaire

Objective

The current usability test has the objective of understanding the main strengths and weaknesses of the current issue creation system.

Participant Characterization

Name:

Department:

Have you ever reported any issue? (Y/N):

Session Explanation

As part of the test, participants are expected to communicate their doubts and lines of thought. This will ensure they are completing the test in a fluent manner and not by trial and error. Additionally, the moderator is expected not to assist the participant during the task and to answer any questions from the participant.

Session Structure

Objectives explanation: 3 minutes

Session explanation: 5 minutes

Tasks development: 30 minutes

Questionnaire: 15 minutes

Total: 53 minutes

Tasks

1. Select the option to create a new issue, and fill in the summary. Confirm that similar issues are presented and click any of them to visualize them;

2. Create a new issue, fill in all fields, attach any attachments, and submit it.

3. Search for the issue that you just submitted;

4. Verify that the issue is presented along with a set of actions to take;

5. Select each one of the actions and fill in the appropriate fields;

6. On the issue that you just submitted and searched, select it to be forwarded to the actual issue in Jira and verify the alterations made;

7. Select the area for information and verify that you are familiar with the content presented.

Questionnaire

Please rate your level of agreement with the following statements. There are 5 levels of agreement:

1-Strongly Disagree | **2-Disagree** | **3-Not Certain** | **4-Agree** | **5-Strongly Agree**

Please mark the corresponding box with your level of agreement in each statement.

Question	**Answer**				
	1	**2**	**3**	**4**	**5**
1- I think that I would like to use this system frequently.					
2- I found the system unnecessarily complex.					
3- I thought the system was easy to use.					
4- I think that I would need the support of a technical person to be able to use this system.					
5- I found the various functions in this system were well integrated.					
6- I thought there was too much inconsistency in this system.					
7- I would imagine that most people would learn to use this system very quickly.					
8- I found the system very cumbersome to use.					
9- I felt very confident using the system.					
10- I needed to learn a lot of things before I could get familiar with this system.					

Comments / Additional Suggestions:

Appendix D

Satisfaction Questionnaire

Please rate your level of agreement with the following statements. There are 5 levels of agreement:

1-Strongly Disagree | **2-Disagree** | **3-Not Certain** | **4-Agree** | **5-Strongly Agree**

Please mark the corresponding box with your level of agreement in each statement.

Question	**Answer**				
	1	**2**	**3**	**4**	**5**
1- Do you consider that the amount of duplicate issues has lowered over time?					
2- Do you consider that the structure of the information improves the task of issue reviewing?					
3- Do you consider that the amount of information gathered improves the task of issue reviewing?					
4- Do you consider that the overall performance in issue reviewing as increased with the implementation of the new system?					
5- Do you consider that the addition of buzz points helps in the management of the visibility of the issues?					
6- Do you consider that the structure of the comments helps in the identification of the different types of comment request?					
7- Do you consider that the information in the issue description is malformed and causes confusion?					

Comments / Additional Suggestions:

Appendix E

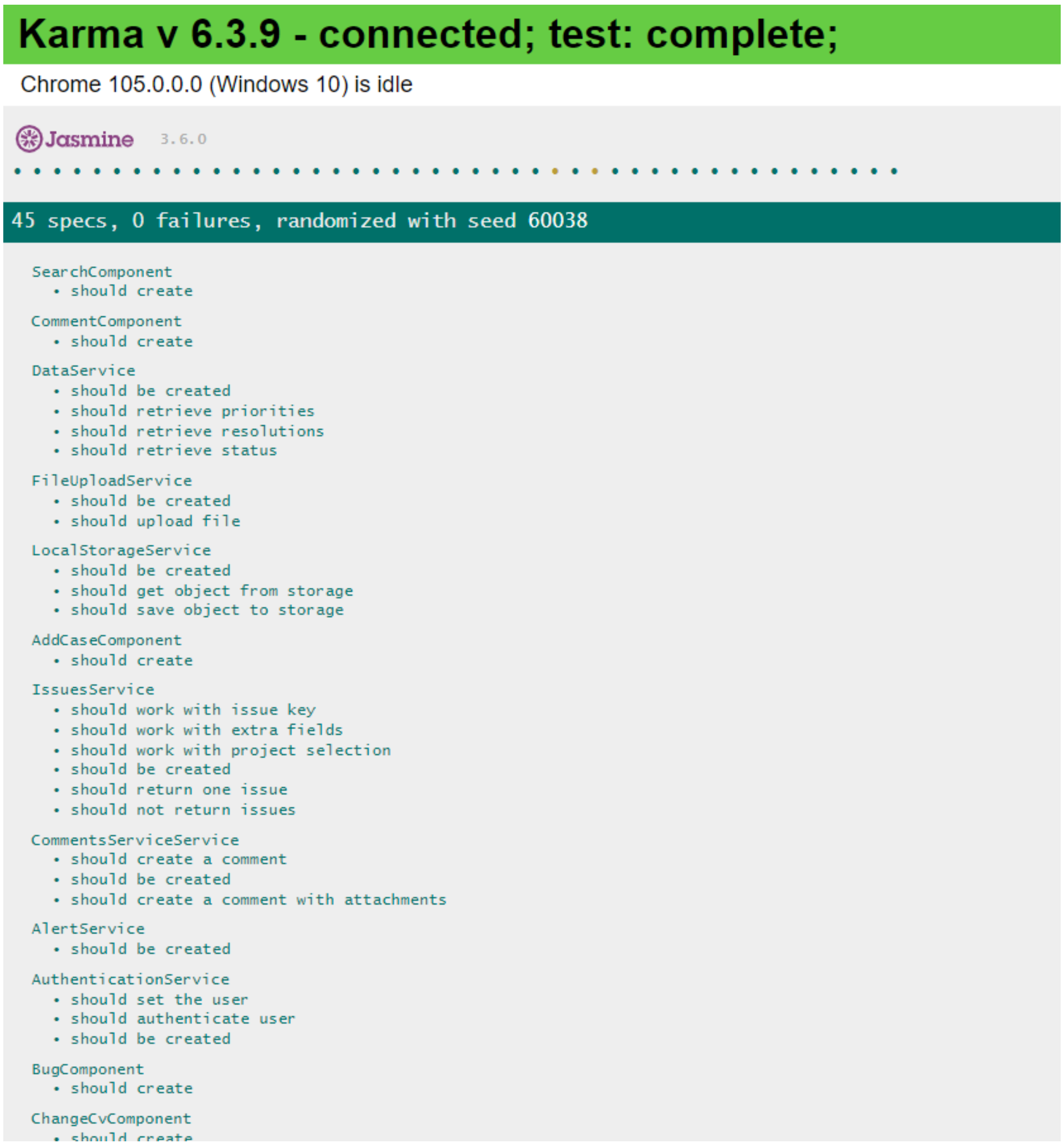
Satisfaction Questionnaire Responses

Results of the Usability Questionnaire. Description: **1-Strongly Disagree** | **2-Disagree** | **3-Not Certain** | **4-Agree** | **5-Strongly Agree**

Department	**Question**										**Final SUS Score**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
Product	5	1	5	1	2	2	5	1	5	1	90
I&D	5	1	5	1	5	1	5	1	5	1	100
Product	5	1	5	1	5	1	5	1	5	1	100
Marketing	3	4	1	3	3	3	3	4	3	3	40
Product	5	2	4	2	4	2	4	2	5	1	82,5
Product	3	3	4	1	3	2	4	2	4	2	70
Product	5	1	5	1	4	1	5	1	5	1	97,5
CEO	4	2	4	1	5	1	4	1	5	1	90
Product	4	2	4	1	4	2	4	1	4	1	82,5
Product	5	1	5	1	5	1	5	1	5	1	100
Product	4	1	4	2	5	1	5	2	4	2	85
SAC	5	2	4	1	4	1	5	2	4	1	87,5
EDS	3	4	3	2	3	2	2	3	4	2	55
Product	5	2	4	3	3	2	5	2	5	1	80
EDS	5	5	4	1	2	2	3	3	4	1	65
Marketing	1	2	4	1	4	1	5	2	4	1	77,5
I&D	5	1	4	1	5	1	4	1	5	1	95
SAC	5	2	5	3	4	1	5	1	4	2	85
Product	5	1	4	1	4	1	5	1	5	1	95

Appendix F

Hypothesis Two - Integration Tests Results



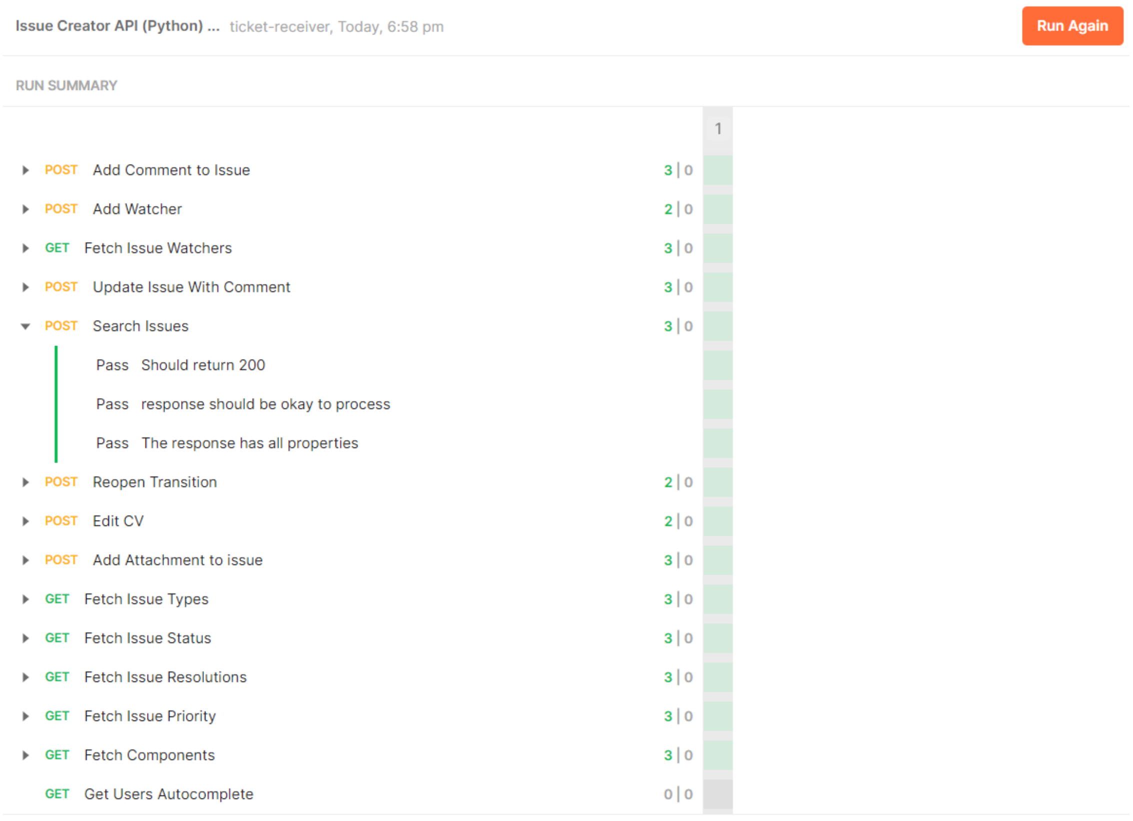
```
Karma v 6.3.9 - connected; test: complete;
Chrome 105.0.0.0 (Windows 10) is idle

Jasmine 3.6.0
•••••••••••••••••••••••••••••••••••••••••••••

45 specs, 0 failures, randomized with seed 60038

SearchComponent
  • should create
CommentComponent
  • should create
DataService
  • should be created
  • should retrieve priorities
  • should retrieve resolutions
  • should retrieve status
FileUploadService
  • should be created
  • should upload file
LocalStorageService
  • should be created
  • should get object from storage
  • should save object to storage
AddCaseComponent
  • should create
IssuesService
  • should work with issue key
  • should work with extra fields
  • should work with project selection
  • should be created
  • should return one issue
  • should not return issues
CommentsServiceService
  • should create a comment
  • should be created
  • should create a comment with attachments
AlertService
  • should be created
AuthenticationService
  • should set the user
  • should authenticate user
  • should be created
BugComponent
  • should create
ChangeCvComponent
  • should create
```

Figure F.1: Integration and unit tests results



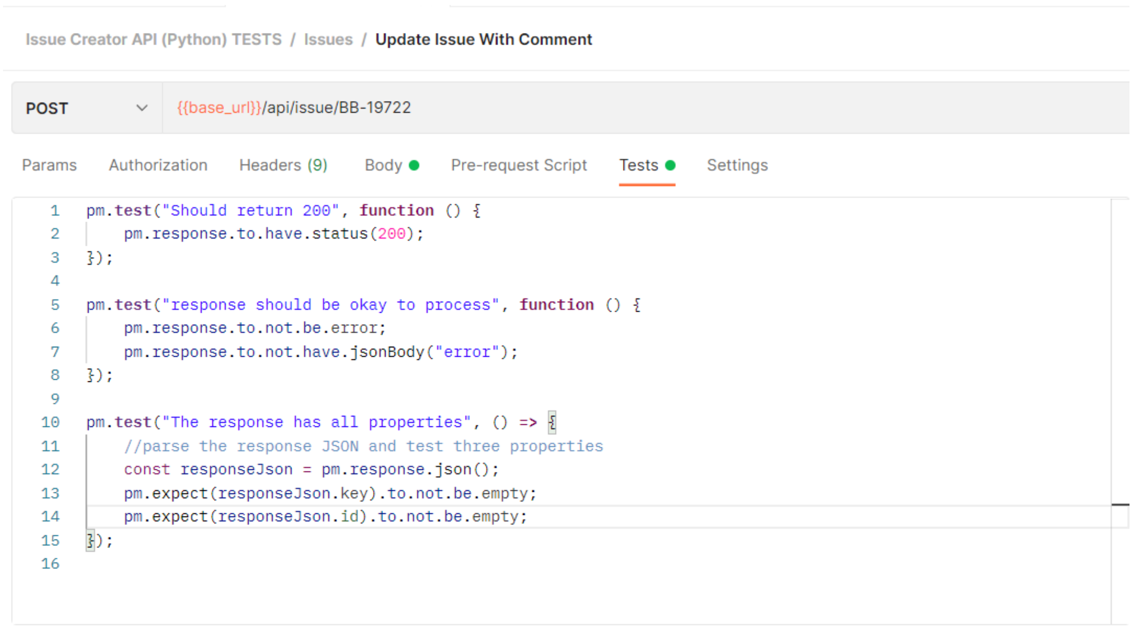
Issue Creator API (Python) ... ticket-receiver, Today, 6:58 pm

Run Again

RUN SUMMARY

		1
POST Add Comment to Issue	3 | 0	🟩
POST Add Watcher	2 | 0	🟩
GET Fetch Issue Watchers	3 | 0	🟩
POST Update Issue With Comment	3 | 0	🟩
POST Search Issues	3 | 0	🟩
Pass Should return 200		🟩
Pass response should be okay to process		🟩
Pass The response has all properties		🟩
POST Reopen Transition	2 | 0	🟩
POST Edit CV	2 | 0	🟩
POST Add Attachment to issue	3 | 0	🟩
GET Fetch Issue Types	3 | 0	🟩
GET Fetch Issue Status	3 | 0	🟩
GET Fetch Issue Resolutions	3 | 0	🟩
GET Fetch Issue Priority	3 | 0	🟩
GET Fetch Components	3 | 0	🟩
GET Get Users Autocomplete	0 | 0	▨

Figure F.2: Integration and unit tests results



Issue Creator API (Python) TESTS / Issues / **Update Issue With Comment**

POST {{base_url}}/api/issue/BB-19722

```
pm.test("Should return 200", function () {
    pm.response.to.have.status(200);
});

pm.test("response should be okay to process", function () {
    pm.response.to.not.be.error;
    pm.response.to.not.have.jsonBody("error");
});

pm.test("The response has all properties", () => {
    //parse the response JSON and test three properties
    const responseJson = pm.response.json();
    pm.expect(responseJson.key).to.not.be.empty;
    pm.expect(responseJson.id).to.not.be.empty;
});

```

Figure F.3: Postman integration test implementation