UNIVERSITAT POLITECNICA DE CATALUNYA

MASTER THESIS

RePlan: Release Planning for Agile development

Author:
Guillem RUFIÁN-TORRELL

Supervisors:
Dr. David AMELLER
Dr. Carles FARRÉ

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Innovation and Research in Informatics

in the

Facultat d'Informatica de Barcelona

June 27, 2016

Declaration of Authorship

I, Guillem RUFIÁN-TORRELL, declare that this thesis titled, "RePlan: Release Planning for Agile development" and most of its contents are original and produced during the master thesis period. However, in some chapters I collaborated with other researchers. Below I explain for each chapter what was my contribution.

All thesis chapters

Dr. David Ameller and Dr. Carles Farré, as supervisors, have contributed to the research described in this study. Both, with the guidance of Dr. Xavier Franch, have been active and involved in the research being conducted, discussing, and writing each paper. Their supervision and global direction has been fundamental for the selection of papers and studies to take ideas to carry this project on.

Chapter 2

This chapter has been co-authorized by Dr. David Ameller, Dr. Carles Farré and Dr. Xavier Franch, all of them professors of the UPC and members of the GESSI team. All three participated in the creation of the State of the Art of this project. Mainly, the first iteration is of their own. Afterwards, the author continued with the following iterations and the final analysis and discussion.

"It had long since come to my attention that people of accomplishment rarely sat back and let things happen to them. They went out and happened to things."

Leonardo da Vinci

Abstract

RePlan: Release Planning for Agile development

by Guillem RUFIÁN-TORRELL

Release Planning methodologies have made possible that project managers and users in general can plan project's releases. These methods try to automatize the human-based planning processes. Currently they are a few web-based and stand-alone tools about release planning, but not all of them offer the same functionalities, like the update of an already planned release or a detailed plan expressed in a timeline. Moreover, these systems are oriented to stakeholders criteria, without taking enough consideration to the available resources. This becomes a limitation, because in many occasions it is vital to have a temporal planning of a release. It also affects key aspects like the planning efficiency or the speed at which it is executed.

In this project a web-based release planning tool has been developed. In this tool, users can create a release with different entities in an easy and simple way. The tool is based in a mathematical model that generates an scheduled plan as tight as possible to the available time and resources. On the other hand, the tool also guarantees the priority fulfillment of features, by respecting the temporal criteria that the user could establish.

The system is also modular, as it can be integrated with other possible different visualizations. Its development in a cloud server also provides public access and scalability.

The tests performed to the system show that the presented mathematical model guarantees the scheduled and efficient planning of a project's release.

Acknowledgements

No voldria continuar endavant en aquesta memòria sense haver mostrat el meu agraïment a un seguit de persones que han contribuït d'una manera o altra en aquesta tesi.
Voldria començar per donar les gràcies al Dr David Ameller i al Dr. Carles Farré, que m'han codirigit aquest projecte i, més que dos directors, hem format un sòlid equip de treball per a poder sol·lucionar totes les dificultats.
No obstant, també li vull dedicar més que les gràcies al Dr. Xavier Franch, que va confiar en mí en tot moment i em va animar a començar, seguir i finalitzar aquesta aventura, l'última de la meva carrera universitària. He d'agrair també tot el suport de l'equip del projecte SUPERSEDE (European Union's ICT 2014 - Information and Communications Technologies Programme), que és l'embrió d'aquesta idea que ha aconseguit sortir endavant.
També vull mostrar el meu reconeixement a les amistats que he mantingut aquests anys, i que he conservat fins al final de la meva educació. Amb especial estima al Roger Jardí, i totes les confidències que ens hem intercanviat aquests últims anys, i al Rodrigo Cueva, company infatigable de batalles fins al final.
Se merecen también todo mi profundo agradecimiento los profesores Antonio J. Nebro y Francisco Chicano, así como Rubén Saboredo, de la Universidad de Málaga. Su puntual pero inestimable ayuda ha sido esencial para poder sacar este proyecto adelante.
Als meus pares, Guillem i M^a Carme, el seu suport, tants dies i tantes nits units davant el cansament, la fatiga i el desig d'abandonar. I tants moments d'unió que ens esperen. Sense ells no hagués tret tota l'energia que tinc per a arribar al final d'aquesta aventura…

Contents

List of Figures

List of Tables

Als meus pares, i al Liam.
Que les seves estrelles marquin el meu camí.

Chapter 1

Introduction

1.1 Outline

The amount of available software services and applications such as mobile apps, web services, etc. has increased exponentially over the past decades [1]. The newest software systems present a higher complexity that the traditional desktop applications, just because they now can be accessed from many devices, and provide smarter features.

Nowadays, the high complexity of real-world software projects justifies the research into computer-assisted tools to properly plan the project development and the flow of its releases. Deciding on the features for an upcoming software release is a high complicated process [2][3]. Moreover, there are time pressures and a limited availability of resources. This ends in having features that are not feasible to implement in a release.

In order to solve these issues, many *release planning* models have been provided. A strong trend consists in deciding which tasks can be performed in a hypothetical next release [4], without considering the scheduling. However, in [5] the authors provide some scheduling techniques without considering the selection of tasks to be done.

Some of the main ideas in this topic have been provided by Ruhe *et al.* [6][7] in many studies, and also in [8], with an online application.

The main issue of the aforementioned ideas is that they do not combine the automation of tasks choice in the next releases and their future scheduling in the timeline. Moreover, the user cannot have control in the process of selecting which tasks can be performed in a release, and also the resources assigned to each task.

Thus, it is necessary to explore another approach to the problem, in order to give the user more control in the process of making decisions. The system should plan the features feasible to be performed taking into account the employees assigned to a concrete project for the upcoming releases.

Moreover, the user should be able to *decide* which features does she want to appear in a concrete release. Up to this point, the decision control is completely transferred to the decision-maker.

The tool should be available for many users to decide in different projects, so it is also required to be published on-line. Therefore, the tool requires of simplicity, usability and classical rules applicable to an on-line application user interface.

Finally, it is required that the tool should be *scalable*. As a project's input can grow gradually over the time, the performance must not be affected dramatically.

1.2 Goals

This study is framed into the SUPERSEDE project [9], whose main objective is to provide tools to support decision-making in the evolution and adaption of software services. One of these tools consists in demonstrating the feasibility to perform the release planning of a software project in the context of agile development using a support system.

Thus, the main outcome of this thesis is to design and implement an on-line release planning system, with the capacity to *select* and *schedule* the features that will appear in the future releases of a project.

This system must provide the decision-maker a set of planning and scheduling operations with her project, by allowing to adjust the desired level of automation.

Moreover, the application must satisfy front-end requirements such as *usability* and *effectiveness*.

1.3 Motivation

The main motivation for the realization of this project is to make a novel contribution to the topic of release planning. Although there is a trending topic in the literature for the last 10 years, we propose a novel alternative which combines both release planning and scheduling algorithms. Finally, the resulting tool has to cover a set of functional and non-functional requirements that make it a public and usable tool.

This contribution is just one among different alternatives that are growing up in the topic of Release Planning, with the objective to make easier the work of project management. In this context, the research and testing of new software adapted to user's decisions should be one the new hypes in the following years.

This study is motivated by the necessity of companies, universities and entities that manage projects to minimize the workload of project management. Thus, the goal is to provide a user-customized tool that plans the different stages of a project in a clear and efficient way.

1.4 Methodology and thesis planning

This thesis has been scheduled to be developed in 18 weeks. The methodology used is based in the Agile paradigms [10]. The main reason of choosing this methodology against others (eg. Waterfalling [11] or Critical Path Method [12]) is the *flexibility* it offers for both the project managers and the developers.

Moreover, the Agile methodology is specialized in increasing productivity and efficiency of software development efforts. In order to build a public tool as in this case, these characteristics are quite desirable.

Taking as basis this methodology, this project has been structured in three main *milestones*, which frame the main functionalities of the tool and allow to run different sprints and iterations during the project's fulfillment. These three milestones are the following ones:

- *Milestone 1*: State of the Art. Research and discussion over the different topics in which this study is framed.
- *Milestone 2*: Deliver of *Plan* version. The tool considers dependencies between features and it can schedule them to the different resources.
- *Milestone 3*: Deliver of RePlan version. The tool allows to modify a plan once it is done with new features. It also considers the features priority as main criteria to make the project's schedule.

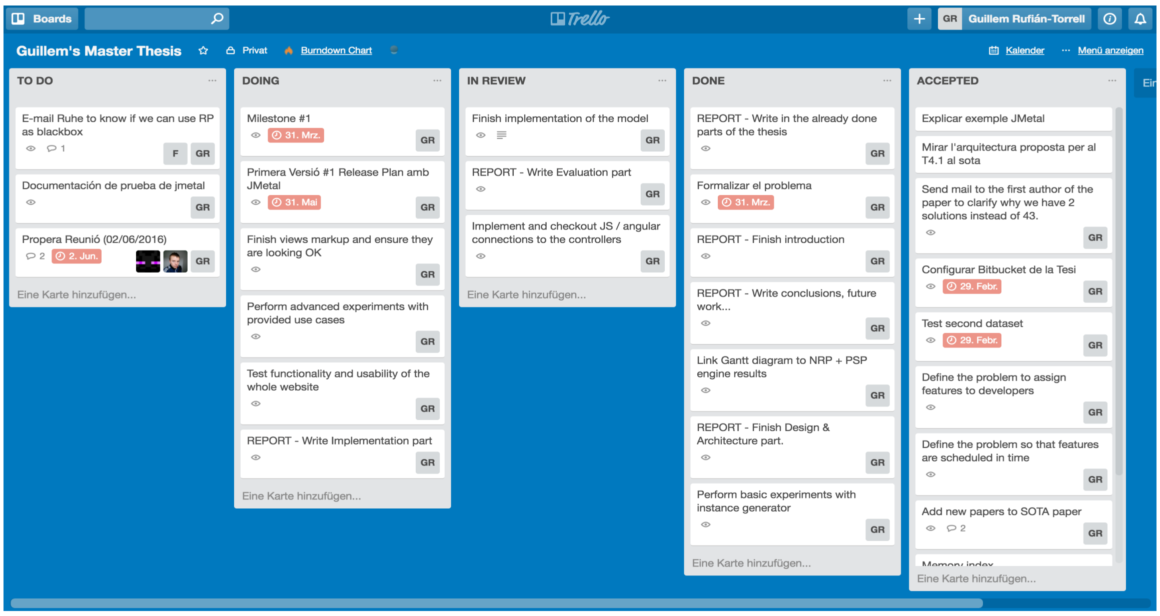
The different deliverables where scheduled in the following time intervals:

Milestone	**Deadline**	**Day**
Milestone 1	Week 2	February 15th
Milestone 2	Week 8	March 31st
Milestone 3	Week 18	May 31st

TABLE 1.1: Scheduled milestones for the RePlan project

Unfortunately, these dates have been prolonged in time for several reasons. The definitive realization of the model or the unbalanced amount of work in Milestones 2 and 3 have been decisive to expand few days the planning of this study.

In any case, the tracking of the whole study has been done by the project managers via Trello [13]. Every task has got different states, depending on the state of fulfillment (see Figure 1.1).



TO DO
- E-mail Ruhe to know if we can use RP as blackbox (1) F GR
- Documentación de prueba de jmetal GR
- Propera Reunió (02/06/2016) (2) 2. Jun. GR
- Eine Karte hinzufügen...

DOING
- Milestone #1 31. Mrz. GR
- Primera Versió #1 Release Plan amb JMetal 31. Mai GR
- Finish views markup and ensure they are looking OK GR
- Perform advanced experiments with provided use cases GR
- Test functionality and usability of the whole website GR
- REPORT - Write Implementation part GR
- Eine Karte hinzufügen...

IN REVIEW
- Finish implementation of the model GR
- REPORT - Write Evaluation part GR
- Implement and checkout JS / angular connections to the controllers GR
- Eine Karte hinzufügen...

DONE
- REPORT - Write in the already done parts of the thesis GR
- Formalizar el problema 31. Mrz. GR
- REPORT - Finish introduction GR
- REPORT - Write conclusions, future work... GR
- Link Gantt diagram to NRP + PSP engine results GR
- REPORT - Finish Design & Architecture part. GR
- Perform basic experiments with instance generator GR
- Eine Karte hinzufügen...

ACCEPTED
- Explicar exemple JMetal
- Mirar l'arquitectura proposta per al T4.1 al sota
- Send mail to the first author of the paper to clarify why we have 2 solutions instead of 43. GR
- Configurar Bitbucket de la Tesi 29. Febr. GR
- Test second dataset 29. Febr. GR
- Define the problem to assign features to developers GR
- Define the problem so that features are scheduled in time GR
- Add new papers to SOTA paper (2) GR
- Memory index
- Eine Karte hinzufügen...

FIGURE 1.1: Backlog of tasks in Trello tool

1.5 Structure of the Thesis

In the following chapters it is explained in-detail the Plan and RePlan system and how have they been implemented. Moreover, this study provides also some case tests applied over the products, and some extracted conclusions. Finally, some future research outlines are also specified.

Section 2 discusses over the existing approaches for Release Planning, by presenting a complete State of the Art. Some iterations are performed in order to take into account all the methodologies presented in this topic. Besides, a background of the topics used is also provided.

The purpose of Section 3 is to define the methodology used in order to accomplish the stated goals of the project. It describes deeply the formulation and converts the requirements into a complete product structure with different layers. The architecture is deeply commented and justified.

Section 4 deals with the implementation of the application products. It gives a complete overview of the technologies performed and describes how to connect all modules and layers in order to convert RePlan into a real-world solution.

The evaluation of the RePlan tool is performed in Section 5. A complete experimentation with the effectiveness of the algorithms is provided, and a sort different real-world case studies are reported. The main goal consists in checking the validity in terms of time-response, usability and clearness of the solutions.

Finally, this study concludes in Section 6 with a short summary of the accomplished objectives, and an outlook for future research.

Chapter 2

Background

2.1 State of the Art

The main objective of this State of the Art is to design a Systematic Literature Mapping with the aim of identifying the main relevant published studies on the topic under consideration. In this case, let us remind that the core of the work is related to Release Planning and software evolution methodologies.

During the experimentation with the possible research queries, a similar study was found in the Literature: the Systematic Literature Review (SLR) by Svahnberg et al. (2010) [14] that investigates the strategic release planning models proposed until 2008, with an particular stress in discussing to what extend they have been validated both in academia and industry. After examining carefully this SLR, the conclusions were that its aims, research questions and results matched the basic needs of this project. Therefore, some decisions in this process have been taken:

1. To use Svahnberg et al. (2010) as a primary source of knowledge about the relevant studies on RP published until 2008. This can be complemented with additional data from Saleem & Shafique (2008), which is the Master Thesis behind the journal version of the SLR.

2. To use Svahnberg et al. (2010) as starting point to find the relevant studies on RP published after 2010, the year of publication of the SLR. The hypothesis that is stated here is that any relevant study on RP published after 2010 should cite the SLR.

3. To use Svanhberg et. al (2010) as a citation reference point. The goal is to find relevant references from other studies that cite explicitly Svahnberg et. al (2010). Those references must cover the period of time from 2008 on. The hypothesis stated here is to find any relevant study on RP in that period of time.

4. To consult a board of experts in release planning. The goal is to find relevant references and books published in the last few years. The final hypothesis here is to find any relevant study or information about RP missed in the other points.

2.1.1 Research Questions

In this study, we formulate the following research questions:

RQ1. What release planning methods have been presented?

This is the most fundamental question of the state of the art. The main goal is to know exactly which are the methodologies in release planning proposed previously in the literature. Still, this question is very broad and then it is decomposed into sub-questions:

- *RQ1.1. What are the main motivations for the methods?* Every method that is formulated may be motivated by different driver(s). It may be related to the quality of the plans, to the effectiveness of the process, etc. With this subquestion the purpose is to identify these factors to be able to check whether they can be applicable to the context of the presented products.

- *RQ1.2. What are the inputs considered by the methods?* Several possible factors may affect the behavior of the method. Typically, any method will need the set of system features to be scheduled with some indication of priority, and some description of available resources, but the details may change.

- *RQ1.3. What are the outputs considered by the method?* Being release planning methods, the primary output should be a planning of releases, being the first possibility the production of just the next release or the update of an existing plan. The attributes bound to releases are also subject of this investigation.

Moreover, another interest in this study is to know the insights of the methods, which motivates the next two research sub-questions.

- *RQ1.4. What are the models managed by the methods?* The definition of models is a central aspect of every method, as it has been reported that expert-based methods are still prevalent over model-based ones (Benestad & Hannay, 2011). This hypothesis must be confirmed in order to learn whether there are some models recurrently used in the surveyed methods.
- *RQ1.5. Which are the algorithms or techniques applied by the methods?* Complementing the former subquestion, the identification of algorithms and techniques used in the methods found is considered of interest given the project aims.

In addition to this first question, it is important to know what has been the relevance of the surveyed approaches and their maturity.

RQ2. To what extent have the release planning methods surveyed in RQ1 been validated?

Similarly as above, this research question has been recomposed into several subquestions:

- *RQ2.1. Are the methods supported by tools?* A fundamental issue about these kind of methods is to know whether they provide a methodological or conceptual framework or instead (or additionally) they are supported by some analysis engine. In particular, the degree of interaction with the analyst (user interaction) needs to be discerned.

- *RQ2.2. How has been industry involved in the methods?* Several factors will be explored in the following sections. For instance, having some practitioners involved in the definition of the methods (by co-authoring publications) is a way to give evidence of industry involvement. Going further, several levels of involvement could be envisaged, being the extreme the case in which a method is effectively used in an industrial process.

- *RQ2.3. What are the major threats identified on the methods?* The main threats reported by each one of the studied methods will be summarized and classified using Wohlin et al.ś classification (2012). A special focus will be taken to generalization, by analyzing possible restrictions of the methods in terms of software process (e.g., Agile), type of organization (e.g., large) or scope (e.g., for commercial package planning, for bespoke systems. . .).

2.1.2 Selection of studies

In order to find the relevant Release Planning methods proposed in the literature since the publication of the SLR of Svahnberg et al. (2010), both forward snowballing and backward snowballing studies have been performed using that SLR as starting point. Snowballing refers to using the reference list of a given paper (backward snowballing) or the citations to the paper (forward snowballing) to identify additional literature (Wohlin, 2014).

It is important to mention that the forward snowballing (first iteration) it is not from the author's own, but for Dr. David Ameller, Dr. Carles Farré and Dr. Xavier Franch, members of the SUPERSEDE project at UPC.

The first forward-snowballing iteration consisted in looking for references citing directly Svahnberg et al. (2010). Since that SLR ends its analysis in 2008, a second backward-snowballing iteration is performed, that is, an analysis of the selected papers references in the first iteration. The goal is to find more RP methods published from 2008 onwards.

For this iteration, both Scopus and Google Scholar were used to find the references that cite directly Svahnberg et al. (2010). 56 references with Scopus, while Google Scholar provided 101. All the references returned by Scopus, except two, were also in the set provided by Google Scholar. Most of the 47 Google Scholar references not included in Scopus were excluded later on (see inclusion/exclusion criteria below), but a few of them were considered. These ones were references to papers published in major journals as online first, without being assigned yet to a concrete journal issue.

The following inclusion criteria is defined, in order to select the relevant studies from the 103 (56+47) papers obtained from Scopus and Google Scholar:

1. The paper is published in:
   - JCR-indexed journal belonging to Q1-Q3 quartiles.
   - Proceedings of one of the following conferences: ICSE, ESEC/FSE, ESEM, HICSS, ICSM, ICSME, and CSMR.
   - Any JCR-indexed journal, CORE A or B conference/workshop proceedings, or book chapter if at least one of the authors has an industry affiliation.
2. The paper describes a release planning method.

A very basic exclusion criterion to simplify the selection of papers has been also defined: all the papers that do not match the 2 inclusion criteria were excluded.

The rationale about the restricted selection of venues in the first criterion is as follows. In this first, and so far only one, iteration of the snowballing procedure, the working hypothesis is that the most relevant studies will be found published in the most renowned journals and venues. The reason for which the selection of journals and venues is relaxed for those studies with authors having an industrial affiliation is that it is desired to find as many as possible studies where a full-scale industrial validation of the proposed RP method is conducted. In the line with the conclusions and recommendations given in Svahnberg et al. (2010), the thought is that industrial validation is a key issue when assessing a RP method.

From the grand total of 103 studies, 81 studies were excluded because they do not match the first inclusion criterion. The titles and abstracts of the remaining 22 studies were read to apply the second inclusion criterion. In the cases where that information was not sufficient to make a decision, the place and context in which Svahnberg et al. (2010) was cited and the full text were also considered (Wohlin, 2014). This resulted in the selection of 9 relevant papers listed in Table 2.1.

Ref	**Year**	**Paper Title**
M1	2012	A hybrid release planning method and its empirical justification.
M2	2011	Quantitative release planning in extreme programming.
M3	2013	Multi-sprint planning and smooth replanning: An optimization model.
M4	2012	Solving the Large Scale Next Release Problem with a Backbone-Based Multilevel Algorithm.
M5	2013	Analyzing an industrial strategic release planning process - A case study at Roche diagnostics.
M6	2013	Continuous release planning in a large-scale scrum development organization at Ericsson.
M7	2015	Software requirements prioritization and selection using linguistic tools and constraint solvers-a controlled experiment.
M8	2014	Industrial evaluation of the impact of quality-driven release planning.
M9	2014	Theme-based product release planning: An analytical approach.

TABLE 2.1: Papers selected from the forward-snowballing iteration.

For the *second iteration*, the papers from the forward-snowballing previous iteration were taken as the starting point and used both Scopus and Google Scholar to find all the references apart from Svahnberg et al. (2010). A total of 647 references were overall selected, without taking into account any inclusion criteria. These references include papers, books, manifestos and programs.

In order to discern the relevant studies from the 647 references, a different inclusion criteria has been established in comparison to the *forward snowballing* iteration.

1. The reference is a paper or a book published after 2008.
2. The paper is published in:
    - JCR-indexed journal belonging to Q1-Q3 quartiles.
    - Proceedings of one of the following conferences: ICSE, ESEC/FSE, ESEM, HICSS, ICSM, ICSME, and CSMR.
    - Any JCR-indexed journal, CORE A or B conference/workshop proceedings, or book chapter if at least one of the authors has an industry affiliation.
3. The paper describes a release planning method.

The exclusion criterion to simplify the selection of papers consists in the following statement: all the papers that do not match the 3 inclusion criteria should be excluded.

The rationale about the restricted selection in terms of publishing year and venues is as follows. In this second iteration of the snowballing procedure, the working hypothesis is that the most relevant studies from 2008 will be found published in the most renowned journals and venues. In line with the conclusions and recommendations given in Svahnberg et al. (2010), industrial validation is a key issue when assessing a release planning method.

From the grand total of 647 studies, 407 studies were excluded because they do not match the first inclusion criterion. The venues and their respective quartiles in which the remaining 240 titles were read to apply the second inclusion criterion. Thus, 115 studies were excluded in this phase of the selection. Finally, the titles and abstracts on the remaining papers were read to apply the third exclusion criterion.

This procedure resulted in the selection of 9 relevant papers listed in Table 2.2.

Ref	**Year**	**Paper Title**
M10	2009	Software project planning for robustness and completion time in the presence of uncertainty using multi-objective search based software engineering
M11	2010	Studying the impact of uncertainty in operational release planning - an integrated method and its initial evaluation
M12	2010	Rigorous support for flexible planning of product release - a stakeholder centric approach and its initial evaluation
M13	2010	Reinforcement learning based approach for adaptive release planning in an agile environment
M14	2011	Conceptual scheduling model and optimized release scheduling for agile environments
M15	2013	The software project scheduling problem: a scalability analysis of multi-objective metaheuristics
M16	2015	Differential evolution with Pareto tournament for the multi-objective next release problem
M17	2010	An integrated approach for requirement selection and scheduling in software release planning
M18	2014	Bi-objective Genetic Search for Release Planning in Support of Themes

TABLE 2.2: Papers selected from the backward-snowballing iteration.

For the *third iteration*, we included few papers indicated by a board of experts in Software Engineering and Release Planning fields. We contacted with recognized authors of the mentioned fields, and after explaining the objectives of our study we asked them to provide valuable references to be included in the study. For the third iteration, a board of experts in Software Engineering and Release Planning fields selected a range of studies about the topic. The experts selected 5 references overall, and all of them were included in this study.

All of them are shown in Table 2.3.

Ref	Year	Paper Title
M19	2016	Risk-aware Multi-Stakeholder Next Release Planning using Multi-Objective Optimization
M20	2015	Next Release Tool
M21	2010	Analytical Product Release Planning (Review)
M22	2010	Product Release Planning - Methods, Tools and Applications (Book)
M23	2010	A study of the bi-objective next release problem

TABLE 2.3: Papers selected from the board of expertsśelection.

2.1.3 Threats to validity

Construct Validity. Selection of primary studies. The selected primary studies have followed a strict protocol. However, the snowballing methodology performed has got some inherent limitations. The most important one is that it narrows the search scope to the referenced papers, therefore many papers may be left out. As a mitigation, the study used as a departing paper is a Systematic Literature Review (SLR), because they are normally cited by many researchers.

The method used is not robust. The protocol may be incomplete, or provide insufficient details. As mitigation we used our experiences in similar studies to provide a detailed protocol.

Internal Validity. Researcher bias. Each paper has been analyzed by the author of this study, however, it is known that it is easy to have different views or interpretations on the same paper depending, e.g., on the research background or past experiences in similar studies. For this reason, the papers were checked by the study's directors when necessary.

External Validity. Results not generalizable. The main purpose of the study is not to be exhaustive, instead it is focused on identifying the most important works on release planning. Therefore, any claim made in this study is limited to the set of studied papers. There may be other release planning methods that were not found in the first iteration. To mitigate this situation, a second iteration using backward snowballing and a third iteration are performed.

Conclusion Validity. Replicability of this study. To allow the replicability of this study it has been defined a precise protocol of the followed steps. However, when using the *forward snowballing* methodology, search engines such as Google Scholar may offer different results in the near future. Therefore, a replication of this study could lead to different selection of primary studies, and in consequence to different results. For the second iteration, the *backward snowballing* methodology has been used, which does not rely on search engines.

2.2 Results

This section summarizes the results of the analysis of the 23 RP references that we have selected (see Tables 2.1 tab:bsnowtab:expertssnow), in relation to the RQs formulated in section 2.2.

2.2.1 RQ1. What release planning methods have been presented?

Sixteen [M2, M3, M4, M6, M7, M10, M11, M13, M14, M15, M16, M17, M18, M19, M20, M23] out of the 23 methods that we have examined propose new methods. The 7 remaining methods [M1, M5, M8, M9, M12, M21 and M22] are extensions to the EVOLVE II method (an extension of EVOLVE) and its implementation as a commercial tool, ReleasePlanner. This latter fact is not surprising due to the high prevalence of the RP methods of the EVOLVE family before 2010 as it is reported in Svahnberg et al. (2010).

RQ1.1. ***What are the main motivations for the methods?***

Among the sixteen new methods, we have identified a first group [M2, M3, M13, M14, M17] whose main concern is to address RP in agile contexts, and then focusing on either tailoring its proposal for eXtreme Programming [M2], or supporting multi-sprint plans [M3, M14, M17], or taking into account previous project's iterations [M13]. A second group of new methods seems more concerned in proposing solutions that scale up in the presence of large sets of requirements. That is the case of [M4, M6, M7, M10, M15, M16].

For the remaining new methods, the motivations differ: addressing the impact of uncertainty with time constraints [M11], supporting weighted criteria from stakeholders [M18], considering different sorts of dependencies between requirements [M19], taking into account the requirements inherent risks [M20] or adding a resource scheduling planning for teams [M23].

For the 7 methods that extend EVOLVE II/ReleasePlanner, the motivations are also different in each case. The resulting extensions, thus, are orthogonal and complementary with respect to the others. In [M1], for example, there is the need of dealing with requirements selection constraints that are more complex and richer than the ones that ReleasePlanner accepted as input. In [M5], the original motivation is to apply ReleasePlanner in an industrial case study and, as a result, a new extension is proposed to address the problem of feature generation. In [M8, M9] the proposed extension of EVOLVE II is driven by the need of dealing with quality aspects and features grouped into themes, respectively. In [M12] there is the necessity to adapt specific stakeholder requirements and constraints and use ReleasePlanner to assign features to different release options. In [M21] the motivation is to define a generic layer to solve different sorts of release planning problems by extracting knowledge and make predictions from large and small datasets. Finally, [M22] is indeed the book in which a complete introduction about release planning and the EVOLVE II method is presented.

RQ1.2. ***What are the inputs considered by the methods?***
In the case of the new methods, the papers provide the necessary information to replicate the analysis of requirements selection factors done in Svahnberg et al. (2010). In Table 2.4 below we present the requirements selection factors addressed by each new RP method. The factors appear in their original formulation and, in parenthesis, their mapping to the taxonomy of requirements selection factors defined in Svahnberg et al. (2010):

- Hard Constraints:
  - Technical Constraints:
    - Requirements Dependencies (RD)
    - Quality Constraints (QC)
    - Other Technical Constraints (TeC)
  - Budget & Cost Constraints (BCC)
  - Resource Constraints (RC)
  - Effort Constraints (EC)
  - Time Constraints (TiC)
- Soft Factors:
  - Stakeholders? Influence Factors (SiF)
  - Value Factors (VF)
  - Risk Factors (RF)
  - Resource Consumption Factors (RCF)

Such constraints, in terms of deadlines for the features, could be required in the Plan and RePlan use cases.

Some of the methods that extend EVOLVE II/ReleasePlanner do not require extra input [M5, M9]. [M1], apart from the inputs that ReleasePlanner requires, is able of accept as input any constraint that can be also expressed as an input of a Constraint Programming Solver. For example, constraint expressing mutual exclusion between features, additive synergy of features, or productivity investments can be considered in [M1]. In the case of [M8], some quality aspects are added to the method. In [M12], the method adds a previous criteria selection from stakeholders.

RQ1.3. What are the outputs considered by the method?

In the case of the new RP methods, ten of them [M2, M4, M6, M13, M16, M17, M18, M19, M20, M23] produce as output the list of stories/requirements/features to be included in the next release. Moreover, in [M2] this list is divided into

		Hard Constraints							Soft Constraints				
		Tech. Constrs.											
Ref	Factors in method	RD	QC	TeC	BCC	RC	EC	TiC	SiF	VF	RF	RCF	Sum
M2	• Story & theme values. • Story sizes + Velocity estimate. • Preference (customers wish) precedences. • Technical precedences.	1					1		1	1			4
M3	• Story value. • Development effort for a story. • Story risk. • Correlation & Precedence among stories.	1					1			1	1		4
M4	• Requirement costs. • Dependencies between requirements. • Which requirements will satisfy the next release to which customers.	1			1				1				3
M6	• Feasibility. • Profitability. • Risk.						1			1	1		3
M7	• Keyword prioritization by user. • Pairwise comparisons between requirements by user.	1				2							3
M10	• Set of available resources/staff. • Set of dependencies between tasks. • Set of different skills.	1				2							3
M11	• Set of efforts for each task. • Set of dependencies between feature's tasks. • Set of developers able to develop release's features. • Set of productivities for a developer to perform tasks.	1				2	1						4
M13	• Set of stakeholders' factors. • Set of requirements dependencies. • Set of effort resources.	1					1		1				3
M14	• Set of resource capacities. • Set of requirements' priorities. • Set of delivery times (for iterations). • Dependencies between features. • Business priorities and efforts (in person/days).	1				1	1	1	1				5
M15	• Salary and maximum dedication of the available employees. • Precedence Graph (TPG) of task dependencies. • Set of different skills.	1				2							3
M16	• Set of weight factor for each client. • Set of costs per requirement. • Set of importance values of reqs for each client.				1				2				3
M17	• Set of available employees organized in teams. • Salary and maximum dedication. • Task dependencies. • Deadline of the project.	1			1	1		1					4
M18	• Set of themes & values per groups of features. • Set of efforts for each task. • Set of dependencies between feature's tasks. • Set of weighted criteria per release and stakeholder. • Set of available resources per release.	1				1	1		1	1			5
M19	• Set of existence dependencies. • Set of cost contributions. • Set of customer value contributions. • Set of temporal precedences. • Set of exclusions.	3			1					1			5
M20	• Set of stakeholders. • Set of requirements' costs. • Set of importances (numeric) that a req has for a stakeholder.				1	1			1				3
M23	• Set of priorities of requirements. • Set of dependencies between requirements. • Set of customers' (stakeholders) weighted relevance.	1							1	1			3

TABLE 2.4: Requirements selection factors per new RP method

three groups with different priority: must-have stories, should-have stories, and could-have stories. In [M17] the output combines the list of next-released requirements with a schedule of the relative time at which these tasks should be performed by the development teams. [M3, M14] produces a multi-sprint plan, i.e. an assignment of stories to consecutive sprints. In [M7], the result is a list of k requirements totally ordered according to their priority (top-k ranked requirements).

On the other hand, other models like [M10, M15] assign different tasks to different developers taking into account the specified constraints. Concretely, in [M11] a time-constraint is added to this output.

The methods that extend EVOLVE II/ReleasePlanner produce the same output than the original method/tool: an assignment of features to the releases in which they have to be implemented.

RQ1.4. What are the models managed by the methods?

A close look to the surveyed approaches show a great diversity of models. First, several approaches adopt a formal approach based on some mathematical formulation [M1, M2, M3, M4, M10, M11, M13, M14, M15, M16, M17, M18, M20, M23]. All of them follow a similar approach to tackle the release planning problem: calculate an assignment from a feature set $f_1, \ldots,$ f_N to a release plan $x = (x_1, \ldots, x_N)$ such $x_j = k$ means that f_j is offered at release k. The solution is required to maximize the value of some utility or objective function. The approaches differ on the details, usually by adding additional information to the utility function.

[M1], [M4], [M10], [M11] and [M15] apply basically this idea. [M1], [M11] and [M15] add a set C of constraints which puts together all possible conditions, e.g. feature priority. [M4] is simpler at this respect since the only constraint explicitly considered in the utility function is budget. In addition, a directed graph records the dependencies between features. [M10] is quite similar to [M1], but it uses a constraint based in the set of developers' skills.

In [M2], due to its application to XP, features are converted into stories and themes that have a business and size (only stories). In order to compute the solution, project velocity is also part of the model. Finally, some decision variables are included. [M13] is quite similar, but it takes as reference the previous iterations and creates a machine learning algorithm to preview the next release.

[M3] is also framed into agile release planning. User stories have utility (business value) and complexity (story points). In addition, velocity, relationships among user stories (coupling, meaning affinity; and precedence) and a measure of risks are added into the function. These factors are modeled using a UML class diagram.

Some similarities are shown in [M14], where features are assigned to the next iteration in an Agile environment. All features are packed in different iterations taking into account requirements dependencies. These factors are modeled using a UML class diagram. In [M16], the model deals completely with stakeholders' criteria, and the main goal is to maximize the value that stakeholders give to different features. Moreover, budget and interactions between task constraints are considered.

[M17] deals with different models, as the goal is to create a scheduled release plan. Anyway, the final model is based on maximizing the revenue provided by each one of the features, considering the time constraints and the available resources.

[M18] is quite similar to other approaches, as the model deals with weighted scores provided by stakeholders to different features. Moreover, the particularity comes with the synergy between features, as the goal is to maximize the appearance of features in the next release that belong to the same generic topic.

In [M19] is framed into the next release problem, with the particularity of applying the satisfiability modulo theorem. The model is based on minimizing cost penalties and maximizing rewards, considering also the contributions of the customer.

[M20] is also framed into the next release problem theory, but the approach is completely different. It deals with a multi-objective model considering the risks provided by the stakeholders and obtaining the least-risky release, taking into account the inputs.

[M23] takes the next release problem with a different perspective, as it uses two objective functions. The goals are to minimize the requirements' cost and maximize the selection of the most critical requirements.

Other model-based approaches are presented in [M5, M9, M12, M21]. In all four cases, they use the ReleasePlanner tool as baseline, therefore it can be said that the underlying model of ReleasePlanner is present in these approaches. In addition, [M5] proposes an extension of Gorschek and Wohlin's RAM model, RAML, that allows linking business strategies with solution planning and development. At its turn, [M9] builds a directed graph to represent feature dependencies (similarly to [M4]). On the other hand, [M12] creates a framework that uses as core the ReleasePlanner tool. The output is not being altered in this case.

[M6], which is based in a large-scale industrial case at Ericsson for agile processes, presents a method which does not use any particular model. In fact, one of the motivations of the case study (in fact, it seems more an action-research initiative) is to overcome the limitations posed by model-based approaches, especially in relation with the assumptions for their application. Therefore, all the release planning is integrated in the traditional agile lightweight agile process.

Finally, two of the approaches, [M7, M8], do not provide much detail on the models used. Both of them share the characteristic that are highly tool-oriented, using some logic-based solvers [M7] and ReleasePlanner [M8], respectively.

RQ1.5. Which are the algorithms or techniques applied by the methods?
Contrary to the previous RQ, we found only two categories of techniques and algorithms applied to the release planning problem.

On the one hand, several approaches [M1, M5, M6, M8, M9, M12, M21, M22] use the ReleasePlanner tool to conduct the release planning problem, therefore the algorithms used are those provided by this tool. The reason for this dominance has clearly to be with the authorship of papers: the eight cited papers are involving Guenther Ruhe, who is the researcher behind the formulation of the EVOLVE family of methods and the ReleasePlanner tool. From these eight approaches, three of them [M5, M6, M8] use the tool only. In the cases [M9, M12, M19], the tool is complemented with some extra functionalities. In [M1], ReleasePlanner is used to generate a first solution that feeds a constraint programming solver to find the best solution with an enlarged set of constraints. Conversely, [M9] uses the output of a graph clustering algorithm to feed ReleasePlanner. In detail, the graph is used, through the interactive Chinese Whispers algorithm, to discover themes by clustering into subgraphs of features (nodes). This is processed by Release-Planner which considers themes as a new kind of relationship, looking for releases that are theme-cohesive. In [M12], Release-Planner is used as the core tool to process the different influences that stakeholders have in the iteration's requirements. Finally, in [M21] ReleasePlanner is also used as a tool inside a 4-layer platform based on maximizing a utility function.

The rest of approaches [M2, M3, M4, M7, M10, M11, M13, M14, M15, M16, M17, M18, M19, M20, M23] build specific solutions by applying powerful algorithms. In [M2], the authors apply a "nested knapsack problem" (which is known to be NP-complete) solver. Releases are considered knapsacks, that are nested because every release includes the preceding one. The problem is to maximize the value that fits into these knapsacks without exceeding their size limits. The concepts and variables mentioned in the previous RQ for [M2] are used by the technique. In their implementation, the solver uses a branch-and-bound algorithm. Somehow similarly, [M4] uses the notion of backbone which is known in algorithm design in constraint solving and combinatorial optimization. The backbone is an ideal structure to model common characteristics of optimal solutions. Given its NP-nature, the authors propose relaxed polynomial forms (approximate backbone and soft backbone; details in [M4]) and blend them into the final combined backbone algorithm proposed in the paper. In [M3], the planning problem is initially converted into a generalized assignment problem, given a linear programming formulation, and solved using a branch-and-cut-based optimizer. In addition, the proposal provides an advanced optimization model that uses a minimum perturbation strategy to ensure stability in case of changes. In [M7], the authors combine several techniques. First, they propose natural language processing in assisting the user in identifying interdependencies and constraints be-tween requirements (this could be considered pre-release planning). The output acts as input for a satisfiability modulo theories solver which produces a first proposal of identification of requirements for the next release, together with a summary of disagreements. These results are used by an analytic hierarchy process to improve the accuracy of the results with human guidance.

In [M10], the authors propose a multi-objective algorithm with a Pareto tournament model, combined with the SPEA II metaheuristic. This means that all the solutions obtained in the population must accomplish a tuple of three objectives to be considered as a solution. The ranking of the best solutions is performed with the Pare-to tournament procedure.

In [M11], the authors combined a Monte-Carlo simulation (to model uncertainty in the operational release planning (ORP) process) with process simulation, as well as an associated optimization heuristic. The method allows for evaluating the impact of uncertainty on make-span.

In [M13], the authors propose a machine learning algorithm that keeps track of the previous iterations, and uses this previous information to predict the next release of the project.

In [M14], the authors developed a multiple knapsack scheduling algorithm, based on several binary knapsack sub-problems. It has similarities with [M2]. It is a branch & bound algorithm, which iteratively selects and schedules an item (feature) for each knapsack sub-problem.

In [M15], the authors apply a multi-objective algorithm and create experiments with different known meta-heuristics in order to compare their performance and results. Then, the approximations obtained from all methods are evaluated with different indicators.

In [M16, M18], the authors use a classical multi-objective algorithm, such as in [M10] or [M15], but they orient it into the Next Release Problem (NRP). The proposal is a new algorithm (Multi-Objective NRP) based in population evolution. All solutions obtained are sorted and evaluated, and the goal is to keep executing the heuristic until all the population members are considered as target solutions.

In [M17], the authors apply a knapsack problem to solve the release planning issue. On the other hand, the authors create a resource-constrained project scheduling problem (RCPSP) in order to schedule the different tasks on time. Then, a combination of both models (in terms of applying more constraints to the formulation) is performed in order to mix both the ideas of release planning with a schedule of the assigned tasks to the developers.

The [M19, M20, M23] cases are also oriented into the Next Release Problem. However, the algorithms are completely different, as the first one uses satisfiability modulo theories to configure their model. The second one uses a more classic multi-objective Pareto front algorithm, just like in [M16] and [M23]. All solutions are evaluated taking into account possible risks.

2.2.2 RQ2. To what extent have the release planning methods surveyed in RQ1 been validated?

To answer RQ2, following we summarize our findings for the three sub-RQs.

RQ2.1. Are the methods supported by tools?

Nearly half of the works found in the state of the art mention some kind of tool, but is worth to differentiate those works that use a tool just to validate their approach (i.e. a prototype or just an ad-hoc solution specific for the paper) from those that are presenting a ready-to-use tool; in this second case, the most remarkable case is Release Planner (this tool was mentioned in

[M1, M5, M8, M9, M12, M21, M22], i.e. all the methods based on EVOLVE-II).

The papers that use a prototype or ad-hoc solution mention the following technologies: CP-Solver [M1], LP-Solve (an OSS linear programming) [M2], and CPLEX [M3]. In general, we can see that all the academic contributions use problem solvers to determine what features will be implemented in the next release.

The rest of papers ([M4, M6, M10, M11, M13, M14, M15, M16, M17, M18, M19, M20]) did not present any kind of tool.

RQ2.2. How has been industry involved in the methods?

All selected papers are academic works (i.e., all or most authors have an academic affiliation). In 5 cases [M5, M6, M8, M10, M11] there was one author from the industry. It's worth noting that these works were the unique that provide real case studies as part of their contribution. The rest of works were validated using experiments with the exception of [M9], which had a case study (using students).

All the proposed methods (except one) were originated in academic research. The exception is an approach proposed by Ericsson [M6], which is also the only one that is being adopted by the industry (by the same company).

RQ2.3. What are the major threats identified on the methods?

We found 5 papers [M5, M6, M7, M8, M10] with a wide analysis of the threats to validity (i.e., including internal, external, construct and conclusion validity threats). In 6 cases [M1, M4, M9, M11, M12, M18] there were some threats explained but only the ones that the authors considered relevant (without organization) and in 11 cases [M2, M3, M13, M14, M15, M16, M17, M19, M20, M21, M22] there was no mention of threats to validity.

2.3 Discussion

A three-phased analysis about Release Planning has been performed in this State of the Art, in order to check the different methods and techniques used during the years. A total of approximately 650 references have been evaluated and either selected or discarded, to come up with the summary table above shown (see Table 2.4).

Thanks to this analysis, we have concluded that the use of the EVOLVE method (and its variants) [M23] is one of the strongest trends in the literature. However, this methodology does not contemplate the scheduling of the release's features assigned to resources.

Moreover, another tendency that can be taken into account is the use of NP-hard problems (such as the Knapsack Problem) to create a Next Release Planning adapted to our necessities.

Finally, the use of original frameworks has also been detected as a way to do, but it can be discarded easily as its design and implementation is too narrow for the RePlan tool's requirements.

However, the first two trends (i.e. the EVOLVE variants and the NP-hard problems) are all implemented with heuristic or metaheuristic algorithms, which is one of the main requirements of our application.

2.4 Study background

2.4.1 Pareto-Optimal Solutions

In order to solve a multi-objective algorithm, many methodologies such as the Multiple Criteria Decision Making (MCDM) and the Evolutionary Multi-Objective Optimization have provided many techniques to solve real problems.

The Evolutionary Multi-Objective Optimization methods (EMO from now on) do not provide a unique solution that optimizes each one of the objective functions. Then, their behaviors affect each other subsequently, and there exists a number of Pareto optimal solutions. Since the decade of

2000 [15] , there has been an increasing tendency to mix the MCDM algorithms and the EMO techniques, in order to introduce the external action of the user (say a project manager) into the EMO methods. The main goal is to safe unnecessary efforts and centralize the focus on the desired solutions, by avoiding the useless ones. There have been lots of approaches of EMO algorithms based on preferences. Some of them modify methods like the Non-Dominated Sorting Genetic Algorithm (NSGAII) to incorporate preferences [16][17] or complement the dominance relationship between the Pareto Solutions.

The planning of software projects has become one of the most intrinsic challenges in the software-developing management. In the last years, with the use of meta-heuristics, the complexity of these problems has accomplished to be affordable.

In this section, some concepts about multi-objective optimization and preferences-based algorithms are introduced.

2.4.2 Multi-objective optimization

A multi-objective optimization problem is mathematically defined as:

In this section, we provide the definition of some concepts for a better understanding of this work. In particular, we define the concept of multi-objective optimization problem (MOP), Pareto dominance and Pareto front. In these definitions we are assuming, without loss of generality, that minimization is the goal for all the objectives.

A general MOP can be formally defined as follows: find a vector $x^\star = [x_1^\star, x_2^\star, ..., x_n^\star]$ which satisfies the m inequality constraints $g_i(x) \geq 0, i = 1, 2, ..., m$, the p equality constraints $h_i(x) = 0, i = 1, 2, ..., p$, and minimizes the vector function $f(x) = [f_1(x), f_2(x), ..., f_k(x)]^T$, where $x = [x_1, x_2, ..., x_n]^T$ is the vector of decision variables.

The set of all values satisfying the constraints defines the feasible region ω and any point $x \in \omega$ is a feasible solution.

Taking into account this definition of a MOP, a solution $x_1 = [x_1, x_{12}, ..., x_{1n}]$ is said to dominate a solution $x_2 = [x_{21}, x_2, ..., x_{2n}]$ if and only if $f_i(x_1) \leq$

$f_i(x_2)$ for $i = 1, 2, ..., m$, and there exist at least one $j(1 \leq j \leq m)$ such that $f_i(x_1) < f_i(x_2)$. Conversely, two points are said to be non-dominated whenever none of them dominates the other. Fig. 1 depicts some examples of dominated and non-dominated solutions.

The solution of a given MOP is usually a set of solutions (referred as Pareto optimal set) satisfying:

- Every two solutions into the set are non-dominated.
- Any other solution, y, is dominated by at least one solution in the set.

The representation of this set in the objective space is referred as Pareto front. Generating Pareto front is the main goal of multi-objective optimization techniques.

In theory, a Pareto front could contain a large number (or even infinitely many) points. In practice, a usable approximate solution will only contain a limited number of them; thus, an important goal is that they should be as close as possible to the exact Pareto front and uniformly spread, otherwise, they would not be very useful to the decision maker. Closeness to the Pareto front ensures that we are dealing with optimal solutions, while a uniform spread of the solutions means that we have made a good exploration of the search space and no regions are left unexplored.

2.4.3 NSGA-II

NSGA-II belongs to the Evolutionary Multi-Objective Algorithms (EMO) [7]. In several studies, such as Zhang et al. [18] and Durillo et al. [4] it is demonstrated that shows better performance and a higher number of obtained Pareto solutions contained in the best fronts in comparison with other similar techniques.

The NSGA-II procedure is shown in Fig. 1. NSGA-II uses an start population P of N candidate solutions (also known as *individuals*. In this algorithm, a combined population $R_t = P_t \cup Q_t$, of size $2N$ is created, where P_t is the original population, and Q_t is the population generated by applying crossover and mutation procedures on the first one. Initially, the original population P_t is generated randomly. Once the two populations are combined, R_t is sorted according to non-domination criteria. Then, solutions belonging to the best non-dominating Pareto-front F_1 in the combined population are chosen. If the size of F_1 is smaller than the number of individuals M, then the remaining members of the new population P_{t+1} are chosen from the subsequent non-dominating fronts. The new population P_{t+1} of size N is now used for selection, crossover and mutation to create a new population Q_{t+1}.

The criteria used to rank the different individuals is very variable and depends on the kind of model used, and the structure of the individuals (or chromosomes). In NSGA-II, the crowded-distance operator evaluates the solutions with highest quality (fitness to the model) and sorts them in descendant order. to select one individual from multiple individuals with differing non-domination ranks, the solution with the lower (better) rank is given preference. If both solutions have the same non-dominating ranks, the one located in a less crowded region has preference.

Algorithm 1 Pseudocode of NSGA-II

```
1: procedure STEPS UP(NSGA-II)
2:     $P \leftarrow$ Initialize Population()
3:     $Q \leftarrow \emptyset$
4:     while not Termination Condition() do
5:         for $i \leftarrow 1$ to nsga-II.popSize / 2 do
6:             parents $\leftarrow$ Selection($P$)
7:             offspring $\leftarrow$ Recombination($nsga - II.Pc, parents$)
8:             offspring $\leftarrow$ Mutation($nsga - II.Pm, offspring$)
9:             Evaluate Fitness($offspring$)
10:            Insert($offspring, Q$)
11:        $R \leftarrow P \cup Q$
12:        Ranking And Crowding($nsga - II, R$)
13:        $P \leftarrow$ Select Best Individuals($nsga - II, R$)
```

The process explained in the previous figure can be done as many times as it is desired (i.e. iterations). When the maximum number of iterations is reached, the *non-termination condition* is fulfilled and the algorithm finishes. Moreover, other parameters, like the size of the population (i.e. the number of generated solutions) can also be set-up.

Chapter 3

RePlan: Design and Architecture of the application

In this chapter there are presented the functional and technical requirements that the scheduled release planner must fulfill in order to work inside the desired environment.

Afterwards, a brief description of the architecture is presented, and a complete formalization of the basic algebraic model of this study. In this model, all the intervening entities and their relationships among them are also introduced.

Then, the different operations that a user can perform in the RePlan tool are introduced and explained: create and manage a plan, and schedule it manually or automatically.

Finally, an in-depth rationale about the RePlan's mathematical model is presented. The goal is to explain the assumed decisions and the process that end up with the main ideas of this model.

3.1 Requirements of the Project

The basic requirements that the application must accomplish are based on the ones established at the beginning of this study. However, some other technical features have been added, in spite of the fact that they affect the structure and implementation of the architecture.

The RePlan's tool requirements are divided into *functional* and *non-functional*.

3.1.1 Functional Requirements

The functional requirements the on-line release planning system must fulfill are based on the smart planning of the project's features inside the subsequent releases. Moreover, it is tempted in any case that all the planned features are the most prioritized ones.

Another fact to remark is that the planning system considers that the projects are following an Agile methodology [10]. Following up this context, all resources assigned to each release are optimized, in order to avoid overtaking the availability of any employee.

Apart from these requirements, the capacity of decision of the project manager has to be guaranteed. Thus, a release can be re-planned at any moment by adding or removing either resources or features.

3.1.2 Non-Functional Requirements

In contrast to the aforementioned functional requirements, the non-functional ones are more dedicated to the *criteria* used to evaluate the implementation and the final use of the RePlan tool.

These criteria is divided into *usability* and *technological* requirements.

Usability Requirements

These requirements are oriented to the usability of the front-end part of the application by the user, which is the final consumer and the main stakeholder of the available operations.

Firstly, all the internal processes performed during the operations should be almost automatic and transparent to the user, executed in the less possible time. The goal is that the user will not notice the complexity of the system by providing her an easy-to-user application.

Moreover, the application's views must be as intuitive as possible, with enough options to guarantee that the user has a complete control about what is she doing with the project.

Technological Requirements

The implementation of an application of this complexity, with such a great amount of entities, requires from a modular language, where different sorts of functions can be engaged. In this case, Java has been chosen because of the main amount of libraries offered, most of them open-source [19].

This programming language allows the implementation of many sorts of algorithms and frameworks. A sub-class of them are the metaheuristic algorithms, necessary to implement the core of this project. Concretely, the jMetal framework [20] has been the election for this thesis. Its open implementation, variety of algorithms and easiness to create new models have been the main reasons of this choice.

Apart from the core, Java also provides libraries for implementing online applications, such as Spring Framework [21], with a high use in the IT industries [22].

3.2 Analysis of RePlan's model

In order to tackle with the requirements of this study, we have designed a problem (i.e. a mathematical model) based on the conclusions extracted from the different proposals provided in Section 2. The goal of this problem is to find solutions that express which features are going to be launched in a release and which resources are dedicated to their fulfillment.

3.2.1 Problem's requirements

The solutions that the appropriate model must find are restricted to a concrete criteria:

- The problem tries to minimize the duration of the release as much as possible, to avoid deadline issues.
- The first features to be launched in the release are the most urgent ones.

- Avoid resources *free time* as much as possible, i.e. all resources must be occupied as much time as possible during the release time.

Moreover, as it has been mentioned before, many features have got dependencies between them. We define these dependencies as pairs $(R_i, R_j) \in A$, where A defines a graph of precedence or succession relationships.

There are different sorts of dependencies that can be dealt into the RePlan tool.

- *Combination*: Feature F_i needs the appearance of feature F_j into the next release, and F_j requires F_i as well. Thus, both features, or none of them, should be included in the next release. This can be expressed as follows in Equations 3.1, 3.2:

$$(F_i, F_j) \in A \tag{3.1}$$

$$(F_j, F_i) \in A \tag{3.2}$$

- *Implication*: Feature F_i needs F_j to be performed. Thus, F_i is only eligible when F_j is also included. This relationship can be expressed in the following way:

$$(F_i, F_j) \in A \tag{3.3}$$

It is assumed that the implication dependency concerns the logical relationship between two features and also their corresponding precedence relation in time. In this study, *only these sort of constraints* are considered.

3.2.2 Natural approach of the model

In order to create a suitable mathematical model that achieves the previous criteria, two kinds of problems recurrent in the literature are considered: decision and scheduling problems.

It is necessary to remark that the complexity of both problems is NP-hard [23, 24]. Therefore, their execution time is very dependent on the input size (e.g. the set of features, dependencies between them, and scheduled resources).

A first approach to recreate this problem is presented by Li et al. in [25]. In this case, their consider the Knapsack Problem [26] as a Next-Release Problem (NRP), and the RCPSP (Resource-Constraint Project Scheduling Problem) [27, 28] to schedule the features launched in the release. A short explanation of the model is explained in the following lines.

For each feature F_j, a binary decision variable x_j is defined, where $x_j = 1$ if and only if the corresponding feature is selected to be launched in the following release. The authors define also v_i as the economical revenue of the feature.

Moreover, the authors consider a *job* entity J_k as the *assignation* of a feature to a concrete resource. Therefore, each feature F_j could have a set of assigned jobs J_k, defined as $X(F_j)$. Finally, for every single job J_k, a group of binary decision variables ζ_{kt} is defined. The possible time interval is $t \in [es_k, ls_k]$, where $\zeta_{kt} = 1$ if and only if job J_k starts at time t.

The objective of Li's model consists in maximizing the revenue of the features included in the release (3.4).

Constraint (3.5) expresses that a feature is selected if and only if all its jobs are planned. Constraints (3.6) and (3.7) deal with the precedence constraints. Constraint (3.6) ensures that a requirement is only selected when its predecessors are selected. Constraint (3.7) guarantees that the jobs for the successor feature can only start after all the jobs for its preceding features are finished. Constraint (3.8) is the resource constraint that one employee is only able to develop one feature at a time. Constraint (3.9) is the binary constraint for all the variables.

$$max \sum_{j=1}^{n} v_j x_j \tag{3.4}$$

Subject to:

$$\sum_{t=es_k}^{t=ls_k} \zeta_{kt} = x_j, \forall J_k \in X(F_j), j = 1, ..., n \tag{3.5}$$

$$x_{j*} \leq x_j \forall (F_j, F_{j*}) \in A \tag{3.6}$$

$$\sum_{t=es_k}^{t=ls_k} t \cdot \zeta_{kt} + d_k \leq \sum_{t=es_k}^{t=ls_{k*}} t \cdot \zeta_{tk*} + (1 - x_{j*} \cdot d(T) \forall (J_k, J_{k*} \in H, J_{k*} \in X(R_{j*})) \tag{3.7}$$

$$\sum_{J_k \in X(G_i)} \sum_{\tau=\sigma(t,k)} t\zeta k\tau \leq 1 \forall t \in 0, 1, ..., T_{max}, i \in 1, ..., m \tag{3.8}$$

$$\zeta_{kt}, x_j \in 0, 1 \forall t \in [es_k, ls_k], J_k \in X'', j \in 1, ..., n \tag{3.9}$$

As this model could fulfill a great part of the aforementioned requirement, its main drawback is that it is considered as an Integer Lineal Problem (ILP). Therefore, as the problem's input size grows, the time to find an optimal solution increments at exponential level. As it has been told before, the main reason for this phenomena is the NP-Hard complexity of this problem.

Moreover, this approach is not able to follow the requirements exposed in Subsection 3.2.1, as this model only deals with a single objective function.

3.2.3 Proposal for RePlan model

In order to avoid these drawbacks, the focus of the study goes to divide this problem into two separated iterations. The first one deals with deciding which features are included in the next release, and the second one schedules the selected features into the project's available resources. Moreover, both problems are multi-objective, so they are able to fulfill the criteria exposed in Subsection 3.2.1.

The selection of features to be launched in a release is accomplished via the Knapsack Problem, which is an optimization problem very present in the literature.

The Knapsack Problem consists in a set of boolean items x_i and numerical weights p_i, so an item is related to a weight. Moreover, every item has also a cost d_i. In the problem that concerns this study, we can extrapolate these variables to the following definitions (see Table 3.1):

Variable	**Definition**
x_i	Boolean variable. Decides the feature's selection.
p_i	Priority of a feature F_i
d_i	Duration of a feature F_i in the timeline.

TABLE 3.1: Definitions for the Knapsack Problem variables in the RePlan tool

The objective consists in selecting the most urgent features to be launched in the release (see Equation 3.10). However, the problem is restricted by limiting the total features duration to the release deadline (Equation 3.11). Moreover, the features appearing in the next release must obey their dependency relationships: a feature cannot appear in a release if its predecessors are not appearing (Equation 3.12).

Thus, considering also the dependencies constraints, the problem can be written as follows:

$$max \sum_{x=1}^{n} p_i * x_i \tag{3.10}$$

Subject to:

$$\sum_{x=1}^{n} d_i * x_i \leq deadline \tag{3.11}$$

$$x_i \leq x_j \, \forall (i, j) \in A \tag{3.12}$$

On the other hand, the schedule of the selected features is in charge of the Project Scheduling Problem (PSP) [29]. It is related to the Resource-Constrained Project Scheduling (RCPS), an existing problem which has been extensively tackled in the literature using both exact techniques [30] and metaheuristic ones [31].

The release's resources are considered as employees with different skills (i.e. capabilities). The employees have a maximum degree of dedication to the project. Formally, each employee is denoted with e_i, where i ranges from 1 to E (number of employees). Let SK be the set of skills, and s_i the ith skill with i going from 1 to the cardinality of SK.

The skills of employee e_i are denoted with $e_i^{skills} \subseteq SK$, and the maximum dedication to the project with e_i^{maxded}. The maximum dedication is a real percentage, which is the ratio between number of hours dedicated to the project and the length of the employee's working day.

Something to remark about the previous lines is that if the skills of a requirement do not match the available employees skills, the feature *will not be scheduled*.

Taking into account the definition of the resources information, and the previously selected release's features F_j ($j = 1, ..., F$), we can define the variables of the scheduling problem.

A solution can be represented with a matrix $X = (x_{ij})$ of size E x F where $x_{ij} \geq 0$. The element x_{ij} is the degree of dedication of employee e_i to the feature F_j.

The objectives of this problem consist in minimizing the duration of the release, reduce the employee's overwork and maximize the effort of the employees in the most urgent feature (see Equations 3.18, 3.19 and 3.20).

Before going on with the expression of all these concepts into equations, we summarize all the variables and information in the following table:

On the right hand, in order to compute the project duration, denoted with p_{dur}, it is essential to calculate the duration of each individual feature F_j (see Equation 3.13). Taking into account that every feature F_j has a *start*

Variable	Definition
E	Representation of the entire set of employees e_i in the project.
SK	Set of skills (or capabilities) of any employee or feature
s_i	Representation of a single skill.
e_i	Representation of a single employee.
e_i^{skills}	Set of employee's skills $s_i \subseteq SK$
e_i^{maxded}	Availability of the employee in the project.
F_j	Representation of a single feature.
F_j^{prior}	Priority of feature F_j.
F_j^{start}	Start time of feature F_j in the timeline.
F_j^{end}	Finish time of feature F_j in the timeline.
X	Matrix E x F of dedications x_{ij} from employees to features.
x_{ij}	Dedication of an employee e_i to a feature F_j.

TABLE 3.2: Definitions for the Scheduling Problem variables in the RePlan tool

(F_j^{start}) and *finish* (F_j^{end}) time, we can define the project duration p_{dur} as the maximum finishing time F_j^{end} found (Equation 3.14).

$$F_j^{dur} = \frac{F_j^{effort}}{\sum_{i=1}^{E} x_{ij}} \tag{3.13}$$

$$p_{dur} = max(F_j^{end}) \, \forall j \in 1, ..., R \tag{3.14}$$

On the other hand, in order to maximize the effort of the employees in the most critical features, we compute the employee's time spent on the project as the sum of the dedication multiplied by the duration of each feature (Equation 3.15).

$$e_i^{work}(t) = \sum_{j | t_j^{start} \leq t \leq t_j^{end}} x_{ij} \tag{3.15}$$

Finally, the overwork of each employee can be calculated using the time schedule of the features and the dedication matrix X, with the following equations:

$$e_i^{over} = \int_{t=0}^{t=p_{dur}} ramp(e_i^{work}(t) - e_i^{maxded})dt \tag{3.16}$$

$$ramp(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{3.17}$$

The problem's objectives are finally expressed in Equations 3.18, 3.19 and 3.20. The constraints express that each feature must be performed by, at least, one employee (see Equation 3.21). The set of required feature's skills must be included in the union of the skills of the employees performing the feature (Equation 3.22), and no employee must exceed her/his maximum dedication to the project.

To sum up all these concepts, let us formalize the problem in the following way:

minimize

$$p_{dur} = max(F_j^{end})\,\forall j \in 1, ..., F \tag{3.18}$$

$$p_{over} = \sum_{i=1}^{E} e_i^{over} \forall i \in 1, ..., E \tag{3.19}$$

maximize

$$x_{i,j} * (F_j^{prior})\,\forall i \in 1, ..., E, j \in 1, ..., F \tag{3.20}$$

Subject to:

$$\sum_{i=1}^{E} x_{ij} > 0\;\forall j \in 1, 2, \ldots, T \tag{3.21}$$

$$t_j^{skills} \subseteq \bigcup_{i|x_{ij}<0} e_i^{skills}\;\forall j \in 1, 2, \ldots, T \tag{3.22}$$

3.3 Architecture of RePlan

The planning system consists into an structure divided into parts, where every one takes responsibility of concrete functionalities. The main goal is to achieve all the functional and non-functional requirements exposed in Section 3.1.1.

The architecture of the RePlan tool can be interpreted in two different ways:

- *Physical architecture*: Definition of the main layers of the application.
- *Logical architecture*: Translation of the physical architecture into a design pattern and the different entities that make the application work.

3.3.1 Physical architecture

The main layers that compose the RePlan tool can be summarized as follows:

- *Planning Engine*: Set of libraries that take charge of the plan in the timeline of a set of project's features and resources. Its modularity makes it available to be used in any environment, either on-line or off-line, by providing a set of operations with their signatures.
- *Database*: Provides persistence to the different entities that interact in the tool.
- *Back-end Server*: WebService in charge of receiving requests from the views, it performs internal and external operations to the database. Moreover, it uses the external engine to generate plans and allows the user to modify them arbitrarialy.
- *Front-end application*: Set of views and underlying logic that translates the information thrown by the Back-end server into visual figures. It is the environment in which the user can interact freely and take decisions over the different entities in her project.

Finally, these components are all connected between them in the following way:

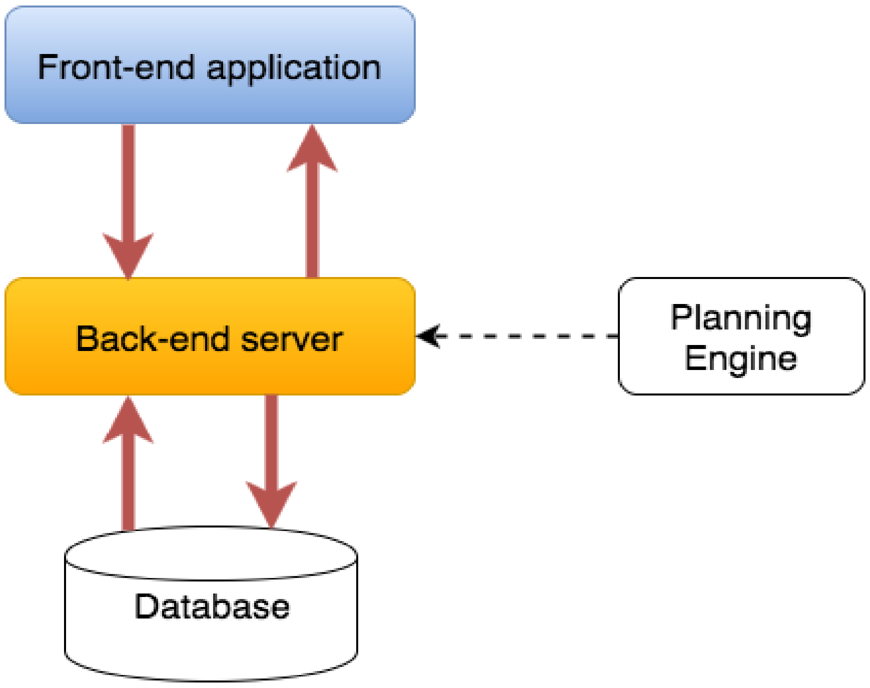


FIGURE 3.1: Physical Architecture of the tool

3.3.2 Logical architecture

The above mentioned components of the RePlan tool can be extended into a more concrete design pattern. Concretely, this tool consists in a *web application* that is built over the Model-View-Controller (MVC from now on) pattern [32]. The main goal is to have a clear differentiation between the *front-end* part and the *back-end* implementation. Moreover, the use of the MVC pattern has been one the most used in both Graphical User Interfaces and Web applications.

As with other software patterns, MVC expresses the "core of the solution" to a problem while allowing it to be adapted for each system. This makes that the different components of the tool are *uncoupled*, so different models, controllers or views can be implemented and extended with relative facility.

The Model-View-Controller structure of the tool is shown in Figure 3.2.

In the following lines, a brief explanation for each part of the pattern is provided:

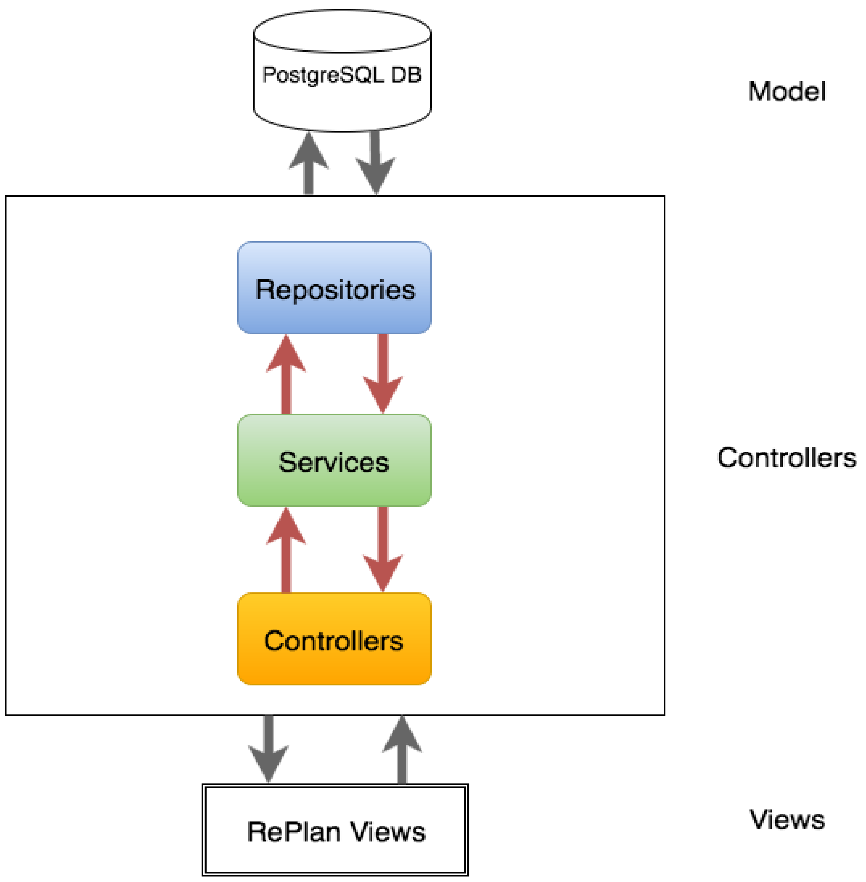


FIGURE 3.2: Structure of the Replan Tool's MVC design

3.3.3 Domain model

The RePlan tool's model is mainly based on the entities mainly described in the different sections of this study. On the right hand, the basic entities necessary to run the planning engine must be included:

- *Release*: Contains a set of features and resources.
- *Feature*: Encapsulates ancestor and successor dependencies, capabilities and textual data.
- *Resource*: It has a set of different *Skills* and availability.
- *Skill*: Represents a capability for both features and resources.

Moreover, in order to allow the user to interact properly with the online tool and make the experience more realistic, some other entities have also been created:

- *User*: Contains basic information about the different users (i.e. username and password). This allows to create properly all the authentication process

- *Project*: A user is able to manage different projects at a certain moment. Thus, also this entity was necessary to be included. It contains a set of *releases* that can be scheduled.

- *Plan*: The key entity of the tool. It contains a set of assignments of features to resources and the starting and ending date for each feature.

The distribution and relationships between all these entities is shown in the UML diagram in Figure 3.3.

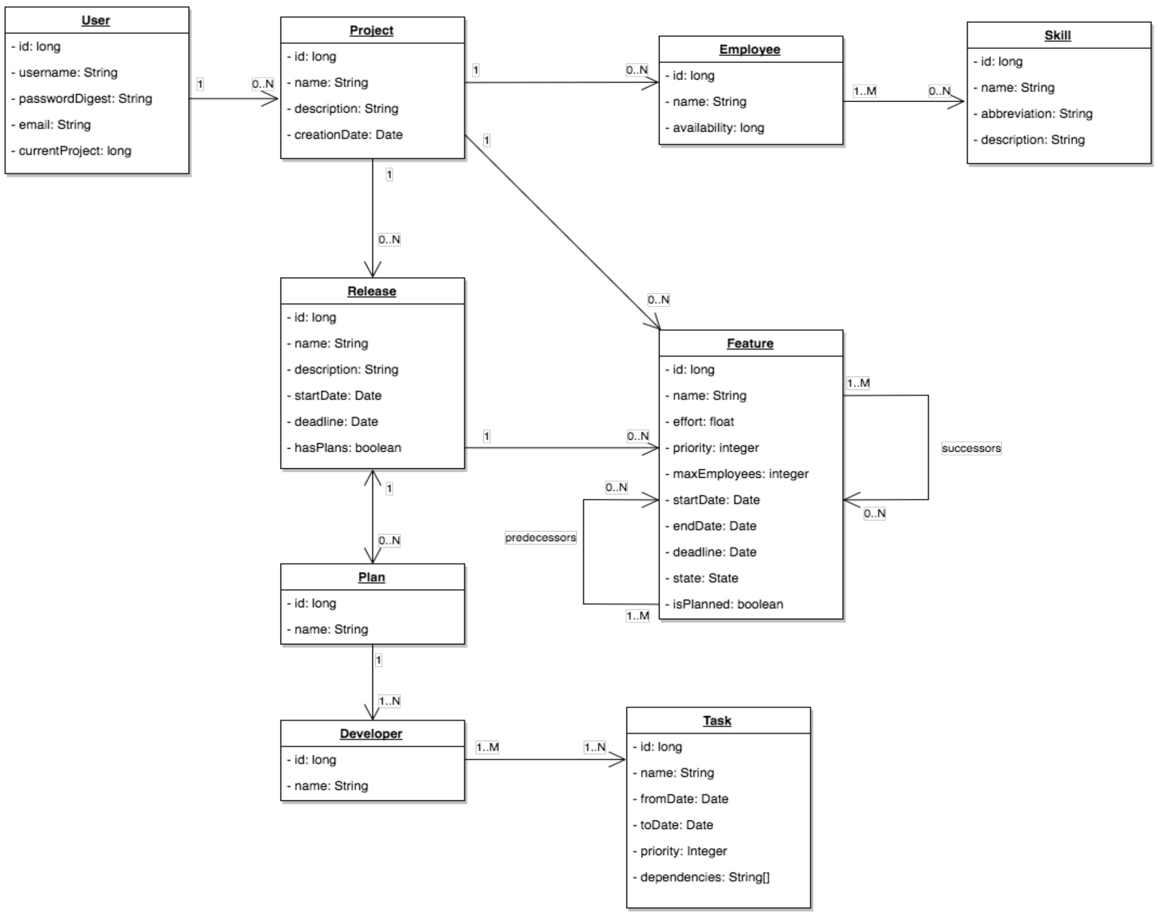


FIGURE 3.3: UML Class Diagram for the RePlan's Domain Model

3.3.4 Controller

The Controller part of the MVC pattern is in charge of all the routing procedures, serialization of data and execution of operations.

As it can be shown in Figure 3.2, the Controller part is actually composed by three sub-layers: the *request-response controllers*, the *services* and the *repositories*.

The set of controllers of RePlan tool are in charge of receiving all the requests from the views and returning adequate response. Among different options to implement this part (i.e. RESTful API, SOAP or other WSDL contracts), the most suitable one is a RESTful API [33].

A RESTful API consists in an architecture of different entities (in this case, controllers), where the focus is on component roles and a specific set of interactions between data elements rather than implementation details [34][35]. Its purpose is to induce performance, scalability, simplicity, modifiability, visibility, portability, and reliability.

An API that adheres to the principles of REST does not require the client to know anything about the structure of the API. Rather, the server needs to provide whatever information the client needs to interact with the service.

At the moment of performing a request by a view, the browser does not know in advance where to submit the information, and it does not know in advance what information to submit. Both forms of information are entirely supplied by the server. This principle can be also referred as HATEOAS [36].

The different operations that can be requested to the RePlan controllers are the classical HTTP ones [37]:

- GET: Retrieval of information identified by the parameters.
- POST: Send serialized information to be saved in the persistence entities.
- DELETE: Deletes the information identified by the parameters.

In order to fulfill the Single Responsibility Principle [38], each one of the running controllers at RePlan takes care of one of the aforementioned entities.

The RePlan tool provides an *external RESTful API* that offers flexibility to the views to obtain the different entities that compose the application.

Each one of the API controllers is defined and described in Appendix A.

When a request is sent to a controller, it automatically receives the parameters and sends them to the *Services* layer.

The *Services* layer is responsible for converting the information received from the controller into entities that can be manageable both for the database and the internal operations of the tool. All entities have got an associated service which performs different conversions depending on the controller's operation (i.e. save an entity, retrieve information, etc.).

In many occasions, the services have to interact with the database. In order to follow a logical decoupling of the tool, it is also necessary to create another layer that is capable to run punctual queries or operations to the database. This is the *Repository* layer.

This layer is in charge of performing all the operations to the database, no matter which kind of controller or API is calling them underneath. Thus, for every one of the main entities acting in the tool there is one *repository* that calls the Create, Retrieve, Update or Delete (CRUD) operations to the database when it is convenient.

3.3.5 Views

This part contains all the visual content that is displayed to the user's browser.

Concretely, in the design of the views we define a *intermediate* layer between the visual document (implemented in a mark-up language) and the RESTful API defined in the previous section.

This layer is composed by a set of *services*. Each one of the services is dedicated to an entity, just in the case of the controllers. The function of each service is to send requests to the API and receive subsequent responses.

Moreover, this layer is accessed from the different views thanks to a set of associated *modules*.

When the user triggers a module's event defined in a view, an operation has to be called inside the module. This module imports different services necessary to process the requests. Thus, it can call to the functionality defined in each service.

Once the service has received the request, depending on the entity it is in charge, it generates the request to the API with the given information. In order to provide more flexibility to the system, the call is done *asynchronously*, so the module can do other operations concurrently. When the call is performed, it receives from the API a set of results, that might be either the design ones or an error message.

The module receives the result from the service, and it performs the convenient operations to pass the data to the view document.

This behavior can be exemplifies in a more technical way via the following sequence UML diagram (Figure 3.4), in which a simulation of a *login* operation is performed.

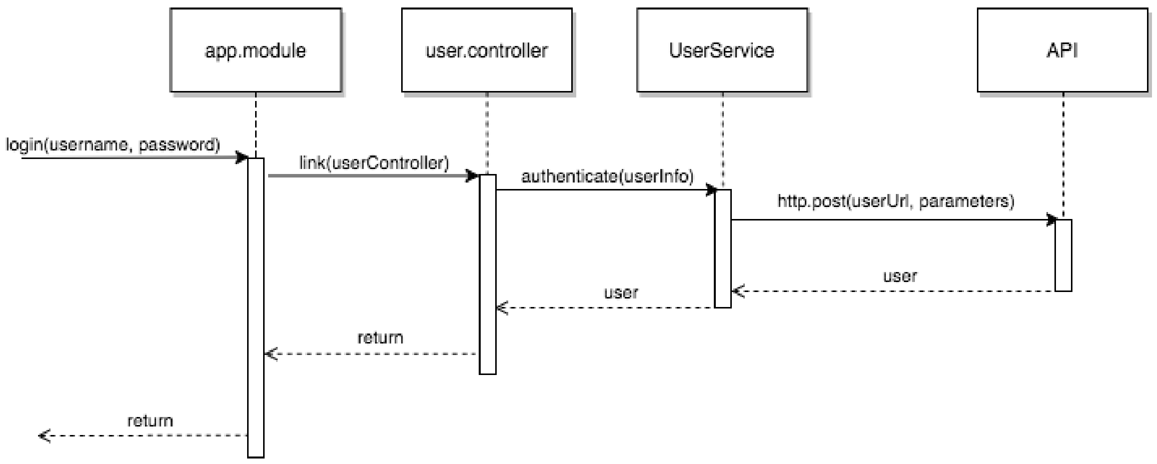


FIGURE 3.4: Sequence diagram of the view's Request/Response pattern

3.4 Functionalities. Operations to perform

The RePlan application consists in a complete application capable to decide and schedule automatically which features are going to be executed in a concrete release. Moreover, it also decides how many resources (i.e. employees) are dedicated to deal with these features.

In the following lines the operations that the user can perform in the RePlan application are described. These operations are able to be performed thanks to the basic engine described in the previous chapter.

First of all, the users should introduce a different sets of project's information into the application. Afterwards, they are few decisions that can be taken depending on the *modality* of planning desired.

3.4.1 Set-up of the Project

The user is allowed to introduce different sorts of information in order to allow the RePlan engine to create a scheduled planning of the project.

First of all, in order to guarantee security in the project manager's operations, the user has to *register* into the application, and then *create a project*. This operation does not present technical complexities, as it is only necessary to introduce some basic data to go forward. The reason of these first steps consists in differentiating information among different projects, as it is more suitable that a company runs several at the same time.

Once in a project, the project manager is able to create different *release* entities rel_i with an optional *deadline* date, in order to restrict which features can be planned inside each of them.

For every release rel_i inserted in the project, the user can assign two different entities:

- A feature F_j with a set of *dependencies* and a set of *abilities*. As it has been introduced in the previous section, each feature differentiates from others depending on their technical characteristics. In the case it has none, a "General" ability is inserted.

- A resource R_k, that can be consider as a company's employee, a client or even external stakeholders. Each resource has also different *abilities* in order to be in charge of different features.

3.4.2 Release Planning

Once the information is set up inside the project, the user is able to decide the modality of release planning. The planning of the features inside the releases can be performed in two different ways:

- *Automatic* : The application is able to assign automatically the sets features in all the releases. It respects also the dependencies between features. This way of execution is simple and fast, as the user cannot do any selection of features or resources. However, it might not produce the desired schedules nor the most effective ones.

- *Manual* : The application gives complete capacity of decision to the user, as she is allowed to select which resources and which features are performed in every release. The set-up process is slightly slow in time, as the decisor must indicate the organization of the features in the releases. However, the plans and schedules produced are completely personalized and dedicated to the user's decisions. This is called the *RePlanning* process.

3.4.3 Features Scheduling

Another point of view for the decisor to take into account is the way she can schedule the features in the different releases.

On the right hand, the RePlan application allows to select *multiple features* and schedule them to a release that has not generated a plan yet. In this case, the user can interact and add or remove features, always respecting the possible *dependencies* of precedence and succession that might exist between them.

On the other hand, as it has been told before, the user is also able to perform the *RePlanning* process, by *adding* new features to the release or *substracting* features that are not in progress to an already scheduled plan. As the projects are running in a real world environment, it is time-illegal to modify any feature that is being executed at the moment or it has been already performed.

3.5 Rationale of the Design

This section is dedicated to give justification to some of the taken decisions in the design of the RePlan application. The following lines mainly deal with the discussion of the chosen model among other many options in the literature.

The selection of a model that suits the necessities of the RePlan application has been one of the most intricate issues of this study. As it is commented in Section 3, the initial idea of the study was to use the problem of Li et al. [25], which combine the Resource-Constrained Project Scheduling Problem (aka. RCPSP) and a classic NP-Hard problem like the Knapsack Problem. However, its impossibility to convert it into a metaheuristic problem and its performance were two major drawbacks to go forward in that direction.

Moreover, an extended analysis was also done over [39], in which Karim et al. present a meta-heuristic multi-objective method to decide which features are going to be released in a hypothetical next release. However, there was no *scheduling* procedures regarding the algorithm. In order to generalize it, the EVOLVE algorithm (and its derivative improves) presented by Ruhe et al [6] do not present a scheduling proposal (eg. RCPSP) to assign features to resources.

Thus, the model turned out into a two-iteration procedure, with the goal to supply response to all the study's requirements. In any case, a multi-objective metaheuristic must be chosen, as the number of objectives to achieve might be variable. In the case of this study, the requirements to fulfill are to minimize the duration of the project, and prioritize the planning of features with more urgency.

The main advantage of a multi-objective metaheuristic is its flexibility to add more objective functions to the model in case of future implementations.

The decision to choose the Project Scheduling Problem among RCPSP or other proposals is based on different reasons.

Firstly, the PSP algorithm can be associated to each one of the employees different attributes that can contribute to the algorithm's objective functions.

Additionally, PSP simplifies the type of resources (in this case, employees) and becomes more suitable to a release planning project. The problem as defined here is more realistic as in other algorithms because it includes the concept of an employee personal skills, also capable of performing several features during a regular working day.

Chapter 4

Implementation of the RePlan tool

In this chapter it is explained, step by step, how all the entities and functionalities in the RePlan tool have been implemented.

First of all, we are going in depth with the implementation of the planning and scheduling engine. This is the base on which the rest of the tool is constructed.

Afterwards, all the entities that take part in the tool are explained. In this case, we are talking about the structure of the RePlan application.

Once all the actors in the application are properly introduced, we are going in depth about the operations performed in RePlan, and how data is transferred between all the entities.

Finally, some comments about the upload of this project to the cloud are provided.

Let us remind that the tool (except the front-end part) is written in JAVA and the libraries used are also developed in this programming language.

All the implementation (both back-end and front-end parts) are available in BitBucket via the URL `http://www.bitbucket.org/guillemon13/replan_website`

4.1 Implementation of the planning engine

The goal of the planning engine is the implementation of a problem that returns a set of solutions. These solutions are *scheduled plans* for a concrete release.

In order to develop all this process, it is necessary to consider an algorithm capable to evaluate the different solutions of this problem and return the most suitable ones with the criteria expressed in Section 3.2. Moreover, it is necessary for this algorithm to be *metaheuristic* in order to avoid overloads of time or efficiency.

As it has been introduced in Section 2.4.3, one of the most used and suitable algorithms to solve this sort of problems is NSGA-II [16], which is included in many metaheuristic frameworks in JAVA [40].

One of the most developed frameworks for the implementation of metaheuristic problems is jMetal [20], presented in 2011 by Durillo et al. JMetal contains a set of algorithms (including NSGA-II, PAES [41] among others).

One of the main advantages of jMetal is that it encapsulates the implementation of all metaheuristic algorithms, so the developer is only required to implement the problem and the criteria to evaluate it.

They are different sorts of problems, depending on the type of the solution variables we want to return:

- Solutions with *double* variables: All the variables are *real* numbers.
- Solutions with *integer* variables. All variables are natural numbers.
- Solutions with *boolean* variables. All variables can take only binary values (i.e. 0 or 1).

For the solution of this model, the variables are expressed in *integer* form, as an employee can be occupied in only *one feature* at a certain moment of time.

The following subsections of this chapter explain in-detail the representation of the data into entities, the implementation of the model and its configuration and the conversion of a feasible solution into a legible scheduled plan.

4.1.1 Representation of data into main entities

As it has been explained in Chapter 3, in this problem we contemplate different entities that contains all the necessary data to be evaluated in order to create feasible solutions.

Let us remind the main attributes that contains each entity:

- *Feature*: It contains a list of ancestor dependent features and successors. Moreover, every feature has a set of capabilities (i.e. *skills*) incorporated. Every feature has got also a *deadline* attribute, which restricts its appearance in the plan.
- *Resource* : It contains a list of capabilities (i.e. skills) and a degree of *availability*. Depending on the availability a feature can be developed in more or less time.
- *Skill* : It contains basic identifying data (i.e. name, description, etc.).

Moreover, for the implementation of this problem, it is also essential to consider a new entity, in order to take advantage of the JAVA modularity and capacity to encapsulate.

Let us consider an entity *Assignation*, which represents that a resource has been assigned to a concrete feature. Logically, it might exist the possibility that a feature contains many assignations. This idea, taken from [25], contributes to an easy and more efficient implementation of the subsequent model.

4.1.2 Implementation of the problem

As it has been commented before, one of the main facilities that the jMetal library offers is the possibility to implement different problem (represented as Problem classes in JAVA) independently from the algorithm used.

In this study, we take profit from this advantage and we implement two iterative (but independent) problems for this model.

As a problem is composed by multiple objective functions and different constraints that must be fulfilled, the implementation in code follows a similar paradigm. A problem in jMetal evaluates independently the objective functions and the constraints of a possible solution.

The RePlan model's is really composed by two iterative problems (the Knapsack Problem and the Project Scheduling Problem). Thus, the implementation of both of them has to be *independent*.

On the right hand, the Knapsack Problem is composed by an objective function (maximize the priority of selected features) and different constraints (the planning of features must not overpass the release's deadline and all features must obey their dependencies).

In order to implement that, the problem receives a set of features as parameter, with the release's deadline. Thus, all evaluations have got the same initial data and no replication is performed.

At the moment of evaluating a solution, the problem computes in the *objective function evaluation* the total priority of the solution's selected attributes. On the other hand, the *constraints evaluation* determine whether there is any feature dependency broken or the duration of the whole release surpasses the deadline.

If one or more constraints have been violated, the problem adds a *penalty* value to the evaluated solution. Thus, a quality criteria can be established and we allow the NSGA-II metaheuristic to *select* the best solutions and *mutate* the worst ones.

When the process has finished, a set of one or more solutions are selected as the best ones. As the underlying algorithm is meta-heuristic, there might be the case that the solution is not the most optimal one. Let us remind that the result of the Knapsack Problem is a set of features feasible to be scheduled in a release.

Now it is the moment to set-up and start the scheduling problem, which presents more complexity in both objective functions and constraints.

Firstly, the set of loaded data is larger, as in this moment we consider both features and resources. Moreover, in order gain efficiency and save replication of operations, before starting the process all the possible *assignations* between features and resources are already created. This is crucial in order to evaluate the solutions, as every one contains as many variables as possible assignations between features and resources.

The reason for this implementation decision is based on gaining efficiency and facility to develop.

The implementation of both problems is based in the same principle: develop both the objective and constraints evaluation function, taking advantage from the facilities that jMetal offers.

On the right hand, the Knapsack Problem creates solutions in which each variable corresponds to the possibility that a feature is selected or not for a release. Its maximization function is developed accumulating the priorities of the selected variables.

Afterwards, the solution is also evaluated in terms of constraints, by detecting whether there is any selected feature with *not selected* predecessors or if there are features that are surpassing the deadline. In negative case, we apply a *penalty* equal to the number of features violating these two conditions.

On the other hand, the Scheduling Problem creates solutions in which each variable corresponds to the dedication of each resource to the previously selected features.

In this case, the evaluation function presents more complexity, as it minimizes the duration of the release as much as possible. Moreover, it also maximizes the most critical selected features execution.

In order to fulfill these conditions, the development is based on working directly with the *assignations* between features and resources. Let us remind that an assignation between a resource and a feature is only considered if the resource's skills contain the totality of the feature's ones.

During the evaluation of the solution, the duration of each one of the features is adapted to the resources degree of dedication provided by the solution variables.

Once all the features duration are adapted, the duration of the release is considered as the maximum end time found over all the selected features.

At the moment of evaluating the constraints, there might be the case that a feature that is feasible to do has no dedication from any of the resources. This is punished with a penalty to the solution.

We have also to consider that there might be the case that an employee is *overloaded* of work. In that case, the duration of the project might be altered as we have to *postpone* the features that cannot be executed at a certain time for this reason. Moreover, for every *overloading* case, we also add a penalty to the solution.

As we can see, the implementation of all the evaluations follows always the same principles.

4.1.3 Representation of the solution

In the following sections, it has been commented that a solution consists in a set of numerical variables. However, these values are not able to express a planning solution as it is shown in the tool.

Each one of the solution variables expresses the degree of dedication of a resource in a concrete feature. In other words, the duration of this feature's development is dependent on the dedication of their assigned resources. Moreover, the start time of a feature's development is logically dependent on its ancestors and the availability of the resources.

Thus, the idea is to encapsulate in the solution the set of *assigned* features to the different resources in the release, with the start and end time of each one in the timeline. The most feasible solutions returned from the NSGA-II algorithm contain all the selected features with their timings, and the only operation left to do is expressing them in the most suitable format (e.g a diagram, textually, etc.).

4.1.4 Final encapsulation

Once all the classes with the planning and scheduling procedures have been implemented, the next step consists in encapsulate it in a unique JAVA library. With this decision, two goals are chased. The library can be imported from any external entity, but without accessing the internal implementation.

Additionally, the jMetal library is also included in order to avoid possible compilation errors.

4.2 Structure of RePlan application

As it has been told in Section 3.1, the RePlan tool consists in a *web application* that is built over the Model-View-Controller pattern [32]. The main goal is to have a clear differentiation between the *front-end* part and the *back-end* implementation.

In the following lines, we will explain how are the different parts of the logical architecture implemented (see Figure 3.2).

4.2.1 Domain model

Taking into account the entities defined in Section 3.3.3, it is clear that in order to persist the information that all the entities contain, some kind of database is needed. For this project, the PostgreSQL Database Management System (DBMS) has been chosen [42] among many other available options.

The main reasons to go ahead with this DBMS are a very high availability and its capacity to be extensible and scalable. These last two features are critical in a tool like RePlan, as the amount of data existing in all projects is *incremental*.

The library used in JAVA to implement the different entities and persist them in the PostgreSQL database is the Hibernate implementation of the Java Persistance API (JPA).

The use of Hibernate obeys to one strong premise: simplicity. Hibernate is based on the JPA main operations, but it also adds useful retrieve operations that gain in efficiency in time and simplicity at the implementation time. Moreover, the fact that JPA is also based in Java Annotations [43] allows to define easily complex relationships between entities (eg. dependencies between features).

Thanks to JPA, any external entity can access easily to any of the entities of the database without writing a SQL query nor performing difficult operations.

4.2.2 Controller - Web Service in Spring Framework

The Controller part of the MVC pattern consists in a Web Service that implements the Spring Framework, which is in charge of all the routing procedures, serialization of data and execution of operations.

Among other functionalities, the Spring Framework features its own requested-based MVC web application framework.

Like Struts, Spring MVC is a request-based framework. The framework defines strategy interfaces for all of the responsibilities that must be handled (e.g. controllers, request-mapping, view resolvers, etc.). The goal of each interface is to be simple and clear, in order to be easy to develop and extensible.

The most important interfaces defined in RePlan and managed by the Spring Framework come as follows:

- *Controller*: comes between Model and View to manage incoming requests and redirect to proper response. It acts as a gate that directs the incoming information. It switches between going into Model or View.
- *HandlerAdapter*: execution of objects that handle incoming requests
- *HandlerMapping*: selecting objects that handle incoming requests based on any attribute or condition.

Among the three interfaces pre-defined by Spring, the key one that centers our attention is the Controller layer. In this layer, we define a set of controllers that are in charge of receiving all the requests from the views and returning adequate response. As it has been defined in Section 3.3.4, all controllers are implemented to a RESTful API design.

Thus, all the operations present in Appendix 1 are implemented following the same structure and signatures. In all cases, when a request is received from an entity's controller, it automatically calls to the service associated, in order to perform the necessary operations to create an appropriate response.

The majority of operations performed by the service are related to Create, Retrieve, Update or Delete (CRUD) operations, which require from an uplying implementation of a model. These operations are all called via the appropriate entity's *repositories*. These repositories are the ones that connect to the DBMS implementation (in this case, a PostgreSQL schema), and perform the necessary operations.

However, there are a set of operations that are the core of the RePlan tool, and they are all associated to the *generation of scheduled plans*. As it has been mentioned before, the RePlan tool extends a set of libraries capable of scheduling and a set of features to a set of resources. This results into a Plan entity, which has to be also saved in the model and *converted* in a suitable way to be redirected in to the appropriate view.

To sum up, Spring framework offers a set of capabilities that manage to handle and map all the request performed by the views, and also all the operations performed to the model, without taking into account their implementation. Thus, we accomplish to follow the Single Responsibility Principle and, what is more important, the cohesion and decoupling from the elements is guaranteed [44], so a new controller, service or repository can be easily extended.

4.2.3 Views

Finally, once explained all the back-end server, it is the moment to comment the implementation of the front-end application.

In general terms, all the views are implemented in HTML5 [45], and contain a main stylesheet (programmed in CSS) with the element's common styles. Moreover, some views punctually include other stylesheet (eg. Bootstrap [46]) in order to set up and organize the views with a usable structure.

However, the most important part of the views are the underlying modules and services that are running behind. They are in charge of performing the requests to the controllers and retrieve data to feed the different view's elements.

All scripts are written in JavaScript, and implement the AngularJS Framework [47]. AngularJS is capable to extend the HTML tags by adding personalized ones, it encapsulates great part of HTML communication and bind the response's data automatically and asynchronously. Its popularity in the last few years is evident, and it has become one of the most used frameworks in front-end programming [48].

Taking into account the combination of these three technologies, all the different views can be implemented with customized functionalities depending on the necessities at each case.

Doing a quick reminder on the design of the views in Section 3.3.5, the election of AngularJS to develop this part of the tool is essential. AngularJS is based on a set of *applications* (i.e. *modules*) that can be associated to each one of the implemented views.

One of these applications can be represented as a *service layer*. This layer implements a set of *services*, each of them capable to perform calls to the RESTful API and the Spring controllers. Each service is dedicated to each one of the entities, and each call creates a request based on an HTTP operation. As it has been told before, each HTTP call is done *asynchronously*, so different requests can be performed to different services concurrently. The main drawback of this configuration are the *dependencies* between information (e.g. features and releases), but it can be solved with a sequential call of the services in some points of the implementation.

Moreover, the rest of the modules implement the necessary services to retrieve the information to show in the views. They perform associated operations that can be callable from the views, acting as an intermediary between them and the front-end's service layer.

4.3 Application operations and communication

The main goal of this chapter is to join and complete the concepts provided in the previous chapter. Moreover, we are dealing with the implementation of the main features of the RePlan tool and the communication between all the entities that take part into every operation.

A very recurrent element in the communications between the controllers and the views are the Data Transfer Objects (DTO). [49]. A DTO consists in a group of different typed attributes that can be transferred from the view to the controller and viceversa. The libraries running in both Spring Framework controllers and the views programmed in Angular parse those DTOs into *JSON* [50] and in the opposite way.

Thus, we explain in the coming subsections how all the operations in the RePlan application work internally, by following the Model-View-Controller pattern.

4.3.1 Set-up of the tool

The first operation the user has to do in the RePlan tool is the *registration*, as it is necessary to have *authorization* to run the different features of RePlan. The registration's view can be seen in Figure 4.1

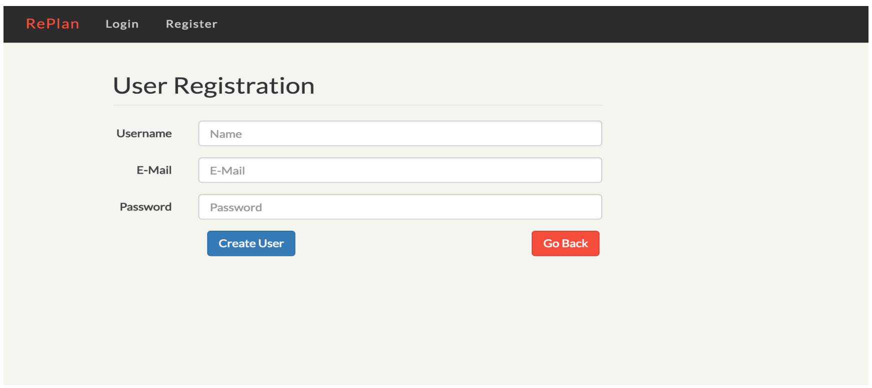


FIGURE 4.1: RePlan tool's registration view

The information (including password) is sent by the user via a DTO to the controller, which sends the data to the service, and reconverts it into a *transactional user entity*. Finally this entity is saved to the database thanks to the repository.

The user automatically is assigned an ID and she can access the tool without doing a login operation.

On the other hand, if the user is already registered, she can login to the tool via a completely different procedure.

4.3.2 Login of the user

The login of the user consists in introducing the *username* and the *password* into the view and submit it to the controller.

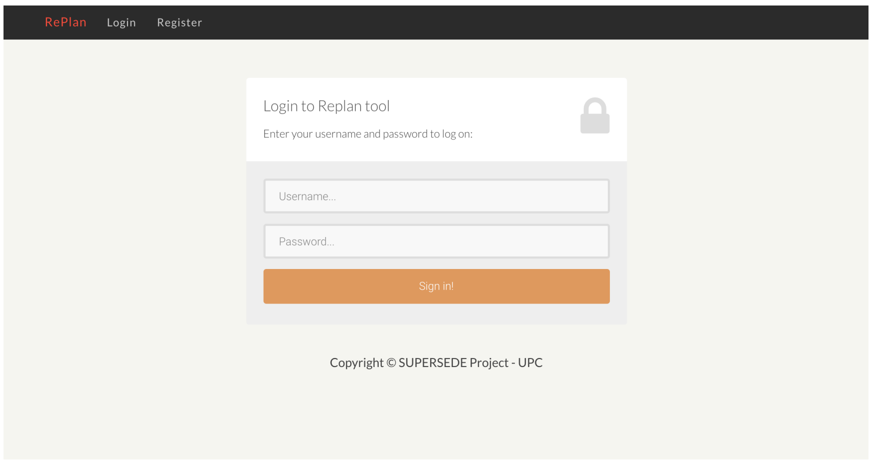

RePlan
Login
Register
Login to Replan tool
Enter your username and password to log on:
Username...
Password...
Sign in!
Copyright © SUPERSEDE Project - UPC

FIGURE 4.2: RePlan tool's login view

The authentication is performed via a *secure asynchronous connection* to an special controller that contrasts the authentication data with the user's information. In case the authentication is successful, the server returns a session *cookie* [51] that allows the user to navigate inside the RePlan's tool.

4.3.3 Project creation

Once the user has accessed to the RePlan tool, the user is obliged to have at least one project in her account. If not, she has to create one in the following form:

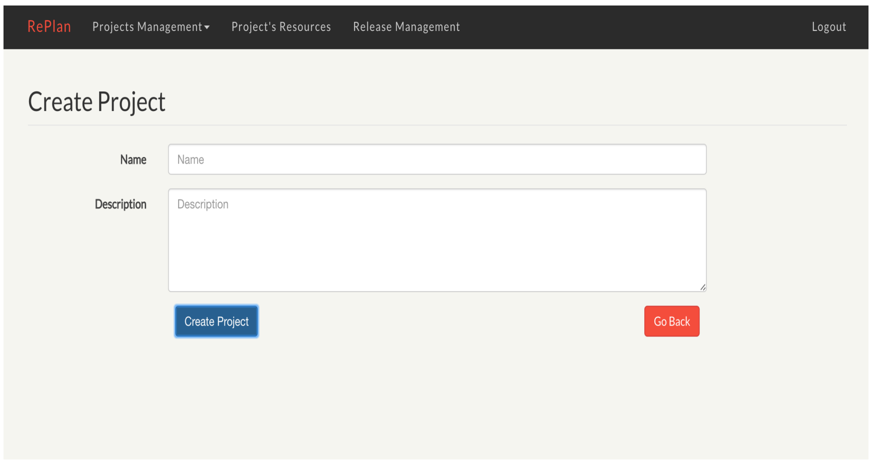


FIGURE 4.3: Replan's tool creation of a project

The view imports an AngularJS script, which is activated with the *Create Project* button. This script mainly collects all the data and creates an appropriate DTO, which is sent to the controller. Internally, the controller dedicated to the projects sends the data to the project's service.

At this point of the operations, two tasks are performed. First of all, the user is updated with the addition of a new project, and, what is more important, the mentioned project is saved via the convenient repository.

As a new project is created, the user now is able to access the Main Screen of the RePlan tool, in order to start managing the available projects.

4.3.4 Main screen

The main screen of RePlan is essentially a two-sided view where the user can *drag and drop* the project's features into the different releases of the project.

As an example situation, we suppose the user has already created some requirements and releases in a new project:

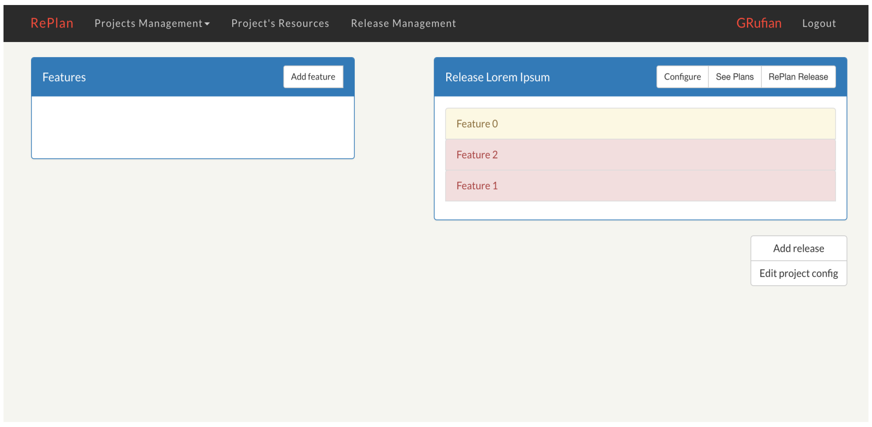


FIGURE 4.4: RePlan tool's Main Screen

Before mentioning all the functionalities of this view, it is important to indicate that many different Angular libraries are imported. The most remarkable ones are *angular-drag-and-drop*, [52] and *angular-messages* [53]. The AngularJS module that this view imports is also extending these libraries.

In the left-side of the view, all the *available features* (i.e. features that have not been planned in any release) are shown. The user is able to *drag* as many features as she wants from this side to any of the releases created in the right-side of the screen.

Moreover, the user can remove *not-planned* features from any release and drag them to the left side of the view or to another release.

One special case to be considered is that if one feature is dragged to the *available features* section, the user will be warned about that fact, and all its successors are also moved.

The implementation behind this functionality is based in the *angular-drag-and-drop* library. It provides both a CSS stylesheet and some methods to allow dragging and dropping elements into different containers.

Moreover, the library also offers an event handling in order to manage via the view's AngularJS module the management of the features in the different sections.

Let us notice that there is any connection to the controller running in this process. Everything is executed via JavaScript and AngularJS scripts.

The main screen view is also the gateway to the rest of the views accessible in this tool. In the left-view section, the user has the option to add or edit a project's feature by accessing the corresponding view. In the right-view section, it is possible to add or edit a release (by accessing its view) or to start the planning process.

Considering that the addition of a requirement consists in just a reference to another view, the release's edition reference and the planning one deserve an special mention.

Firstly, the release's edition is a reference to a *parametrized view*, i.e., it needs the ID of a release to make the underlying script to show its data. In order to avoid showing the ID of the release in the URL for privacy reasons [54], we have decided to implement the AngularJS routing [55] in the view's module. The routing library allows to asynchronously load different templates without reloading the page nor changing the public URL.

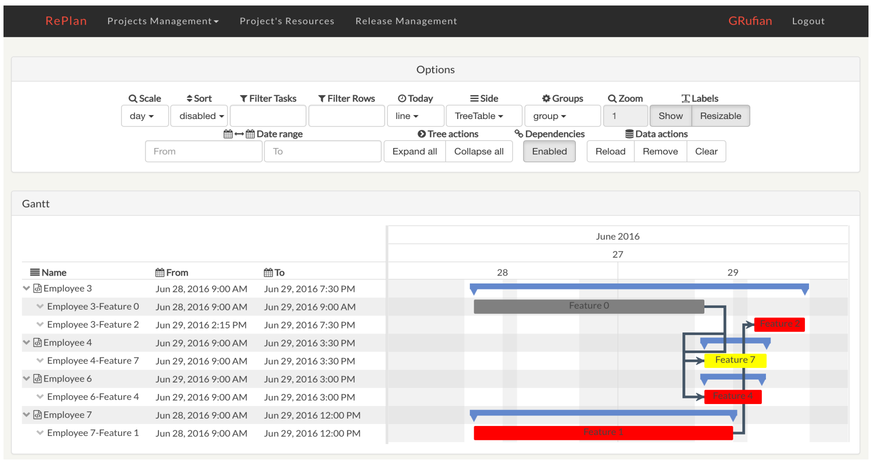
On the other hand, the 'Confirm' button recollects all the requirements existing in the release's container and updates them to the corresponding controller via the main screen's module.

Once the release is updated into the database, the module invokes the planning controller in order to *generate* a plan (or set of plans) for the release's requirements and resources.

While the RePlan engine is creating the plan solutions, the tool will stay active. At the point everything is ready, the tool redirects automatically to the planning view.

4.3.5 Planning view

In this view, a complete Gantt chart [56] represents the scheduled plan of the release's features and the resources assigned to their performance. This chart is mostly implemented with the AngularJS Gantt Diagram [57].



Scale	Sort	Filter Tasks	Filter Rows	Today	Side	Groups	Zoom	Labels
day	disabled			line	TreeTable	group	1	Show / Resizable

Date range		Dependencies
From	To	Enabled

Name	From	To
Employee 3	Jun 28, 2016 9:00 AM	Jun 29, 2016 7:30 PM
Employee 3-Feature 0	Jun 28, 2016 9:00 AM	Jun 29, 2016 9:00 AM
Employee 3-Feature 2	Jun 29, 2016 2:15 PM	Jun 29, 2016 7:30 PM
Employee 4	Jun 29, 2016 9:00 AM	Jun 29, 2016 3:30 PM
Employee 4-Feature 7	Jun 29, 2016 9:00 AM	Jun 29, 2016 3:30 PM
Employee 6	Jun 29, 2016 9:00 AM	Jun 29, 2016 3:00 PM
Employee 6-Feature 4	Jun 29, 2016 9:00 AM	Jun 29, 2016 3:00 PM
Employee 7	Jun 28, 2016 9:00 AM	Jun 29, 2016 12:00 PM
Employee 7-Feature 1	Jun 28, 2016 9:00 AM	Jun 29, 2016 12:00 PM

FIGURE 4.5: RePlan Tool's example of a Release Plan

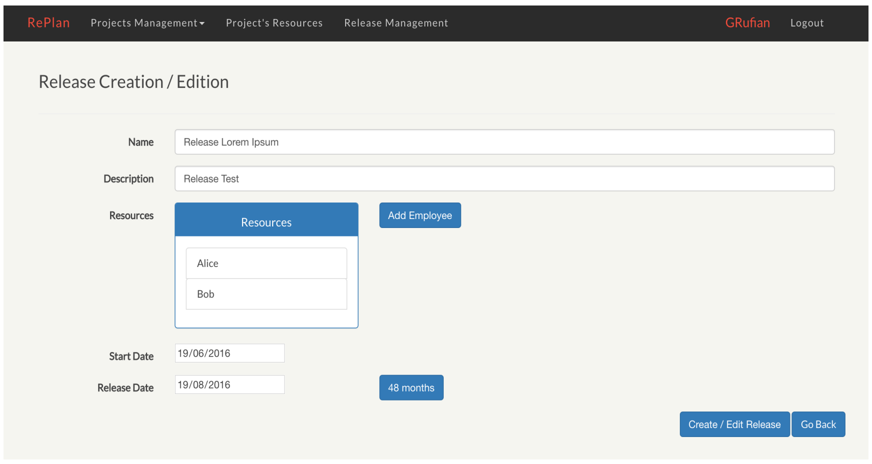
As it can be seen, the diagram offers many options for the user to *filter* the diagram's data and adapt it to her necessities. Some of the filtering that the user can do come as follows:

- Change the date scope to zoom data with more or less time precision.
- Get a subset of features from a date interval.

These functionalities are all encapsulated in the Angular-Gantt library, which offers an internal API to be called when necessary.

4.3.6 Releases creation and edition

From the main screen, the user is also able to create or edit the data of a release. At this point, a personalized view for every release is shown as follows.



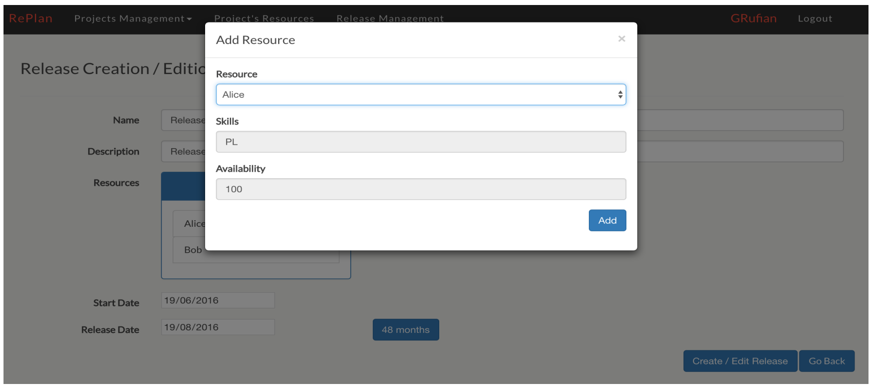
Release Creation / Edition

Name	Release Lorem Ipsum
Description	Release Test
Resources	Alice, Bob
Start Date	19/06/2016
Release Date	19/08/2016

FIGURE 4.6: RePlan tool's release setup view

As it can be seen, the user can enter different sorts of textual data (eg. name, description, planned start and end). Moreover, the most important feature of the view is the *resources section*.

The user can add a resource by clicking to the *Add Resource* button. Automatically, a modal dialog [cite:modaldialog] appears with the available resources (i.e. employees) of the project that have not been inserted yet to the release.



Resource: Alice
Skills: PL
Availability: 100

Name: Release
Description: Release
Resources: Alice, Bob
Start Date: 19/06/2016
Release Date: 19/08/2016

FIGURE 4.7: RePlan tool's release's employee addition modal

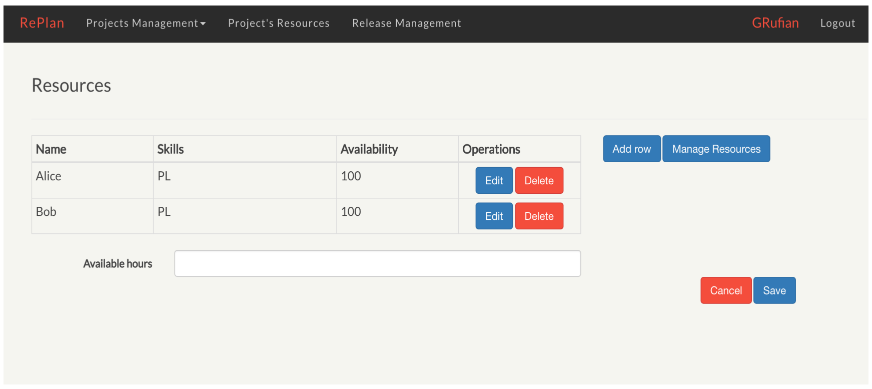
Once the user selects and accepts a resource into the release, it can be shown in the *resources section*. If at any moment the resource has to be deleted from this release, we have only to drag it out from the list to any point of the screen.

Let us remind that this resources drag-and-drop functionality is based o the same principle as in the Main Screen's view. In this case, the Angular module only contemplates a list of resources that can be manipulated either directly from the view or from other Angular methods.

Once all the changes have been done, the release can be saved and sent to the appropriate controller.

4.3.7 Resources management

This view takes charge of the whole project's resources (i.e. employees) and their subsequent capabilities (i.e. skills). The resources are shown in the following table:



Resources

Name	Skills	Availability	Operations
Alice	PL	100	Edit Delete
Bob	PL	100	Edit Delete

Available hours

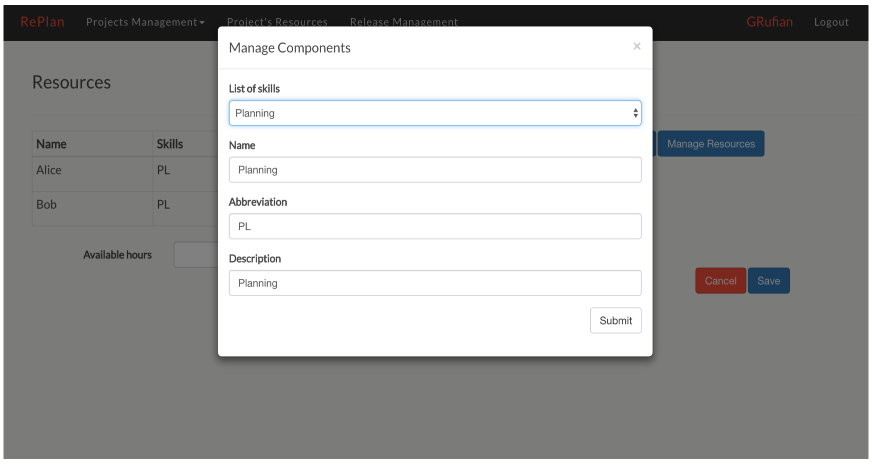
FIGURE 4.8: RePlan tool's resources setup view

As it can be seen, each row represents an *employee*, with a set of *skills* and an availability percentage. The user has the option to either *edit* or *delete* any of the resources in the table, or *add* new ones via the Add Resource button.

In any case, the edition of a resource is composed by two textual inputs and one *multi-selection* input for the capabilities. This last input detects automatically which skills has every employee and shows a list of the still available skills to choose. The changes can be either saved or canceled at any moment.

All these operations call to underlying methods in the view's AngularJS module.

On the other hand, the available project skills can be modified in a modal dialog. Moreover, also new skills can be added to the project.



Manage Components

List of skills: Planning

Name: Planning

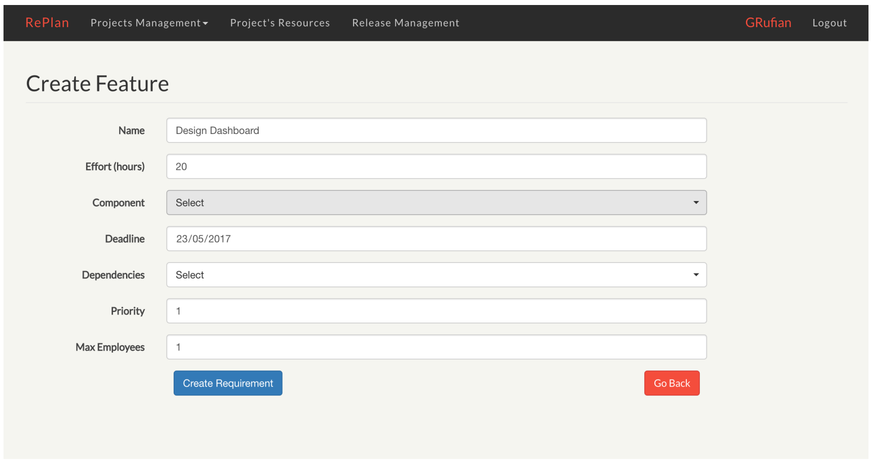
Abbreviation: PL

Description: Planning

FIGURE 4.9: RePlan tool's skills setup view

4.3.8 Features edition

A feature can be created at any moment of the tool's execution or edited by selecting one from the Main Screen. Anyway, the feature's edition view come as follows.



Name	Design Dashboard
Effort (hours)	20
Component	Select
Deadline	23/05/2017
Dependencies	Select
Priority	1
Max Employees	1

FIGURE 4.10: RePlan tool's feature's setup view

It is composed by a set of textual fields and two *multi-selection* lists [cite:multiselect] in the *capabilities* (i.e. skills) and the *dependencies* selection. In both cases, it is more than probable that a feature has more than one dependency and more than one skill to be loaded.

At the moment to be sent to the controller, these two attributes are considered as current lists of DTOs. The rest of textual data is encapsulated directly to the feature's DTO.

4.4 Compilation and upload application to the cloud

The compilation and testing of the application has been performed thanks to Maven [58], which is a build manager from Apache. Maven is capable to import external libraries and plug-ins and compile them to the project automatically (eg. the Hibernate API or the own jMetal library).

Another of the crucial plugins to add to the project is the Tomcat 7 plug-in. This allows to compile the whole application and *publish* it in a Tomcat [59] container.

The Tomcat container can be *local* (for testing) or *remote*. In the second case, we might choose a provider that allows us to publish both our tool and the PostgreSQL database.

For this tool we chose Heroku [60] as our provider for both the application and the database. Heroku offers a free plan adapted to our necessities.

The upload to this server is also done with maven, which builds and pushes automatically the project to the Heroku repository.

The application can be accessible via the URL

`http://replanupc.herokuapp.com`

Chapter 5

Evaluation

In order to evaluate the different functionalities that the RePlan tool provides to the user, a complete incremental evaluation has been performed. The evaluation of this study is mainly divided into two parts:

- *Basic tests*, checking that the restrictions of the model are being fulfilled.
- *Instance tests* with sets of resources, features and skills predefined for the RePlan scheduler engine. In this case, the efficiency is the parameter to take into account.
- *Global tests* of the whole online tool with different execution alternatives. Here, efficiency and usability meet to establish an objective evaluation.

Finally, the discussion about the tool's efficiency is provided, an compared with other algorithms.

5.1 Basic tests

In order to evaluate the different restrictions that our model presents, let us suppose a set of critical cases:

- *Priority dominance*: The first features to be executed are the most urgent ones.

- *Dependency restrictions*: Some features have precedence restrictions, so they cannot be scheduled if any ancestor of them has not been selected.
- *Capability restrictions*: A feature cannot be planned if there is no employee capable (i.e. with enough skills) to completely perform it.

It is important to remark that the planning is simulated as in the real world. Thus, the duration of the features is dependent on the working daily hours, so it might be the case that a feature can be extended up to more than one day.

5.1.1 Priority dominance tests

For this test, we consider a release with a set of three features F_1, F_2 and F_3, with respective priorities and duration. Moreover, this release also has two employees assigned (E_1 and E_2). In order to ensure that this test is only priority-dedicated, we assign the same skills to all features and resources.

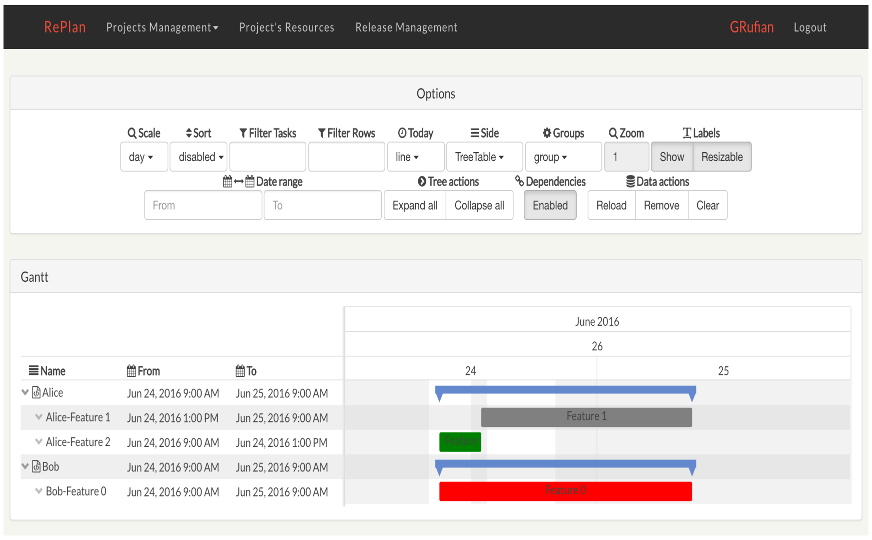
The definition of all three features is shown in Table 5.1

Feature	**Priority**	**Duration**
F_1	2	5 hours
F_2	3	7 hours
F_3	1	7 hours

TABLE 5.1: Definition of features for the priority dominance test

Taking into account the definition of the model, we are trying to minimize as much as possible the *duration* of the release. Moreover, the realization of the features must obey a priority order.

The results of this test can be shown in Figure 5.1



Scale: day; Sort: disabled; Today: line; Side: TreeTable; Groups: group; Zoom: 1; Labels: Show, Resizable; Dependencies: Enabled

Name	From	To
Alice	Jun 24, 2016 9:00 AM	Jun 25, 2016 9:00 AM
Alice-Feature 1	Jun 24, 2016 1:00 PM	Jun 25, 2016 9:00 AM
Alice-Feature 2	Jun 24, 2016 9:00 AM	Jun 24, 2016 1:00 PM
Bob	Jun 24, 2016 9:00 AM	Jun 25, 2016 9:00 AM
Bob-Feature 0	Jun 24, 2016 9:00 AM	Jun 25, 2016 9:00 AM

FIGURE 5.1: Priority dominance test results

As it can be shown, features F_1 and F_3 are executed firstly, as their priorities are higher than F_2. Thus, feature F_2 is postponed until one of to employees is free to carry it out.

5.1.2 Dependency restriction tests

In this test, let us define a 5-featured release, with 3 employees. As in the other case, in order to avoid external influences, let us suppose all features and resources have got the same capabilities.

The definition of the features and their dependencies is shown in Table 5.2 and in the following *graph of dependencies* (Figure 5.2).

Feature	**Ancestors**	**Duration**
F_1	-	7 hours
F_2	F_1	5 hours
F_3	F_1, F_2	4 hours
F_4	-	5 hours
F_5	F_2	4 hour

TABLE 5.2: Definition of dependencies graph for subsequent test

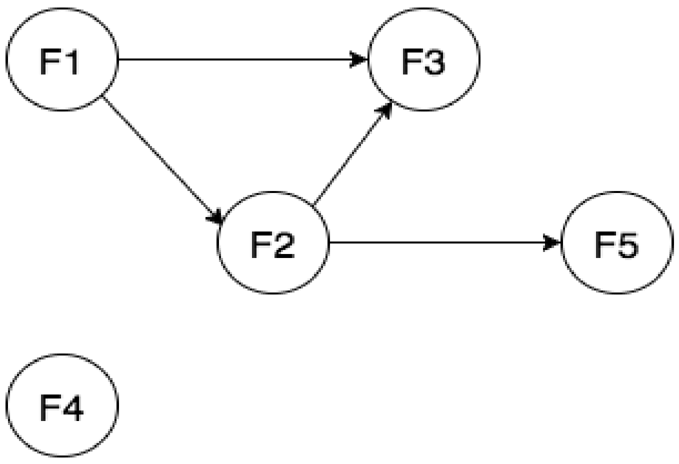


FIGURE 5.2: Graph of dependencies for dependency restriction test

The scheduled plan is shown in Figure 5.3.

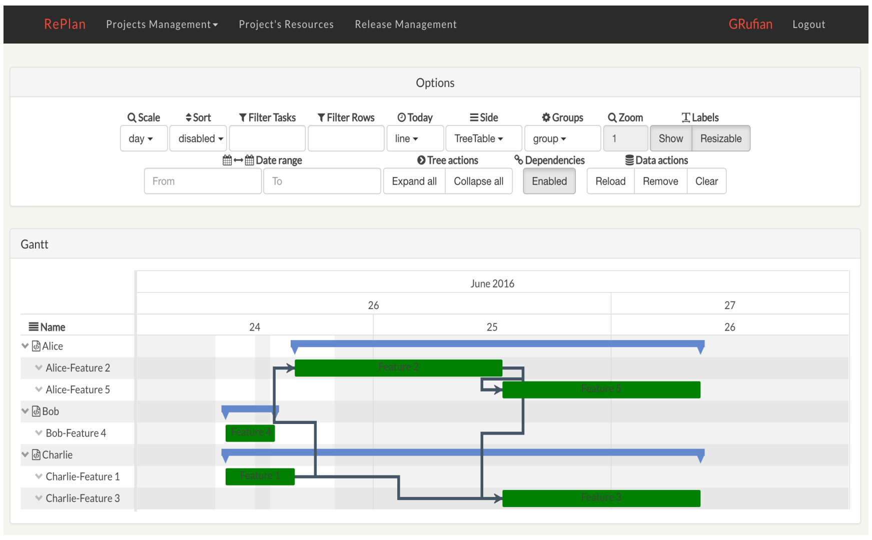


FIGURE 5.3: Replan's Dependencies test results

As it can be seen clearly, all features obey their restrictions and are exactly executed at the time their ancestors finish.

5.1.3 Capability restrictions

The last restriction of the model to test is the scheduling of features that are *feasible* to do by any employee at any time inside the planned release dates. In this case, it might occur that some features cannot be done because the release's resources do not have all the necessary skills to perform them.

In Tables 5.3 and 5.4 features and resources capabilities are defined.

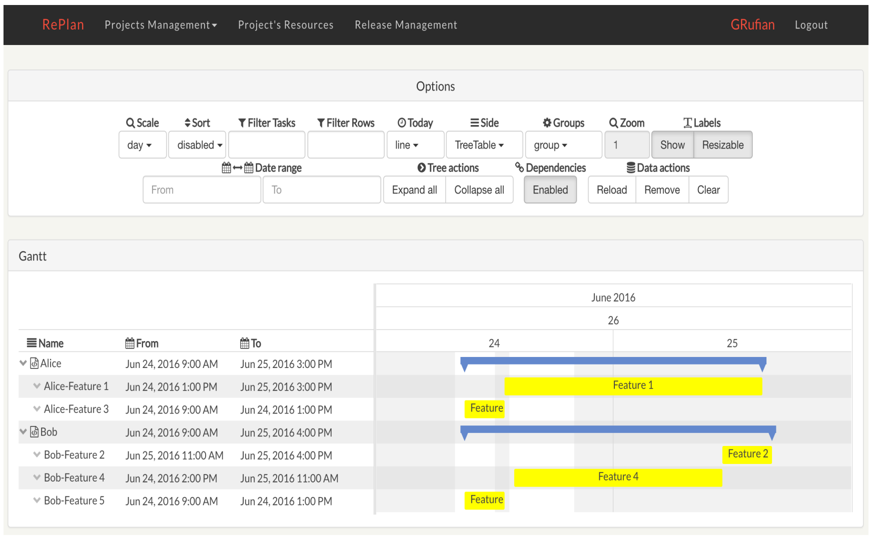
Feature	Skills	Duration
F_1	Sk_1, Sk_2, Sk_3	8 hours
F_2	SK_1	4 hours
F_3	SK_2	3 hours
F_4	SK_2 SK_3	6 hours
F_5	SK_1	2 hours

TABLE 5.3: Features skill definition for the capabilities restriction test

Resource	Skills
R_1	Sk_1, Sk_2, Sk_3
R_2	SK_1
R_3	SK_3
R_4	SK_1, SK_2
R_5	SK_1

TABLE 5.4: Resources definition for the capabilities restriction test

The results of this scheduled plan are shown in Figure 5.4



Name	From	To
Alice	Jun 24, 2016 9:00 AM	Jun 25, 2016 3:00 PM
Alice-Feature 1	Jun 24, 2016 1:00 PM	Jun 25, 2016 3:00 PM
Alice-Feature 3	Jun 24, 2016 9:00 AM	Jun 24, 2016 1:00 PM
Bob	Jun 24, 2016 9:00 AM	Jun 25, 2016 4:00 PM
Bob-Feature 2	Jun 25, 2016 11:00 AM	Jun 25, 2016 4:00 PM
Bob-Feature 4	Jun 24, 2016 2:00 PM	Jun 25, 2016 11:00 AM
Bob-Feature 5	Jun 24, 2016 9:00 AM	Jun 24, 2016 1:00 PM

FIGURE 5.4: Replan's Skill test results

As it can be seen, features have been excluded from the planning as their skills do not match with any of the selected resources' ones. Therefore, resources *Alice* and *Bob* do have to perform all features, as the rest are not able to deal with them.

5.1.4 Testing all three restrictions. Real cases.

Last but not least, we are going to show two more tests that are oriented to show real instances of execution by mixing the three constraints analyzed individually before.

Test 1.- Unfeasible plan

In Tables 5.5 and 5.6 we define a list features with different skills and duration, and a list of available resources with their corresponding skills. Taking into account the amount of inputs and the complexity of the dependencies, we can consider this a test as a proof for a *real scenario*.

Feature	Skills	Ancestors	Priority	Duration
F_0	Sk_3, Sk_4, Sk_5	-	1	9 hours
F_1	Sk_4, Sk_3, Sk_6	-	4	12 hours
F_2	Sk_0, Sk_7, Sk_3	F_0	0	11 hours
F_3	Sk_7, Sk_6, Sk_0	F_0, F_1	3	12 hours
F_4	Sk_1, Sk_0	F_3	2	7 hours
F_5	Sk_0, Sk_7, Sk_5	F_2, F_3	1	4 hours
F_6	Sk_3, Sk_5	F_2	2	17 hours
F_7	Sk_6, Sk_1, Sk_4	F_5	4	12 hours
F_8	Sk_6, Sk_9, Sk_2	F_7, F_1	2	5 hours
F_9	Sk_0, Sk_4, Sk_2	F_4	2	9 hours

TABLE 5.5: Features skill definition for Real Case Test 1

Resource	Skills	Availability
R_0	Sk_2, Sk_6, Sk_7	100%
R_1	Sk_8, Sk_0, Sk_1	100%
R_2	Sk_5, Sk_3, Sk_9	100%
R_3	Sk_6, Sk_0, Sk_3	100%
R_4	Sk_5, Sk_4	100%

TABLE 5.6: Resources definition for Real Case Test 1

Before going more in depth with this case, let us notice that there are some features that *cannot be performed* by any employee as they do not match in their capabilities (see Table 5.7).

Thus, as some unfeasible features have dependencies, their successors also cannot be performed (see the *graph* in Figure 5.5).

Feature	Skills	Feasibility
F_0	Sk_3, Sk_4, Sk_5	Unfeasible
F_1	Sk_4, Sk_3, Sk_6	Unfeasible
F_2	Sk_0, Sk_7, Sk_3	Unfeasible
F_3	Sk_7, Sk_6, Sk_0	Unfeasible
F_4	Sk_1, Sk_0	Feasible
F_5	Sk_0, Sk_7, Sk_5	Unfeasible
F_6	Sk_3, Sk_5	Unfeasible
F_7	Sk_6, Sk_1, Sk_4	Unfeasible
F_8	Sk_6, Sk_9, Sk_2	Unfeasible
F_9	Sk_0, Sk_4, Sk_2	Unfeasible

TABLE 5.7: Features feasibility for Real Case Test 1

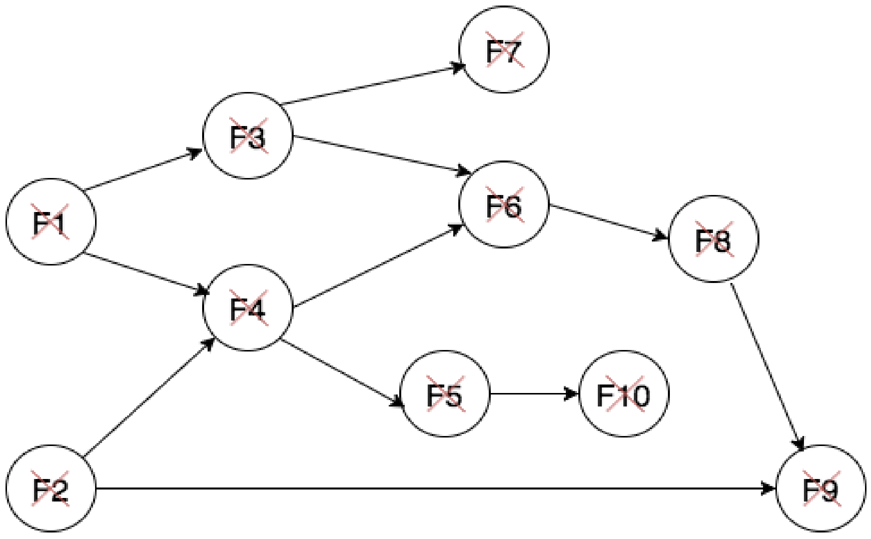
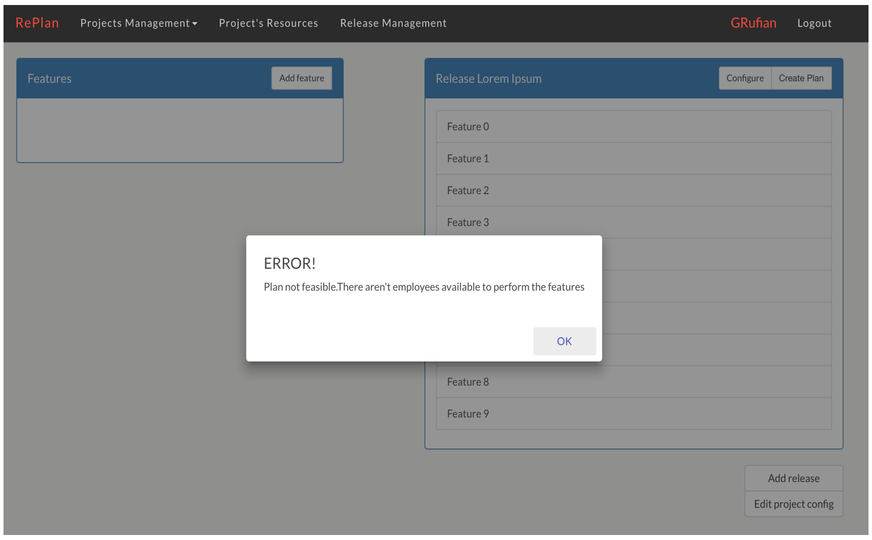


FIGURE 5.5: Replan's Skill test results

RePlan will throw a message explaining there is no possibility to generate a plan with this combination of features, employees and capabilities.



ERROR!

Plan not feasible.There aren't employees available to perform the features

FIGURE 5.6: Replan's Unfeasibility Real Case Test 1

Test 2.- Feasible Plan

Let us show now in Tables 5.8 and 5.9 a combination of features and resources that do match in some features, so the RePlan generator can create one or more release plans (see Figure 5.8)

Feature	**Skills**	**Ancestors**	**Priority**	**Duration**
F_0	Sk_7, Sk_8	-	2	16 hours
F_1	Sk_2, Sk_7	-	2	22 hours
F_2	Sk_3, Sk_6	F_1	2	8 hours
F_3	Sk_0, Sk_7, Sk_9	-	4	23 hours
F_4	Sk_5, Sk_7	F_0	1	11 hours
F_5	Sk_4, Sk_5, Sk_9	F_2, F_3	1	15 hours
F_6	Sk_4, Sk_5	F_2, F_3	3	6 hours
F_7	Sk_3, Sk_5	F_0	4	4 hours
F_8	Sk_1, Sk_9	F_2, F_3	1	12 hours
F_9	Sk_1, Sk_3, Sk_8	F_1, F_8	4	9 hours

TABLE 5.8: Features skill definition for Real Case Test 2

Resource	Skills	Availability
R_0	Sk_1, Sk_3, Sk_5, Sk_8	100%
R_1	Sk_1, Sk_5, Sk_6, Sk_8	100%
R_2	Sk_2, Sk_3, Sk_4, Sk_7, Sk_8	100%
R_3	Sk_3, Sk_5, Sk_6, Sk_7, Sk_8	100%
R_4	Sk_3, Sk_4, Sk_5, Sk_9	100%
R_5	Sk_0, Sk_1, Sk_2, Sk_5	100%
R_6	Sk_2, Sk_5, Sk_7, Sk_8	100%
R_7	Sk_1, Sk_2, Sk_3, Sk_6, Sk_7	100%
R_8	Sk_1, Sk_2, Sk_3, Sk_5, Sk_7	100%
R_9	Sk_0, Sk_2, Sk_4, Sk_5, Sk_8	100%

TABLE 5.9: Resources definition for Real Case Test 2

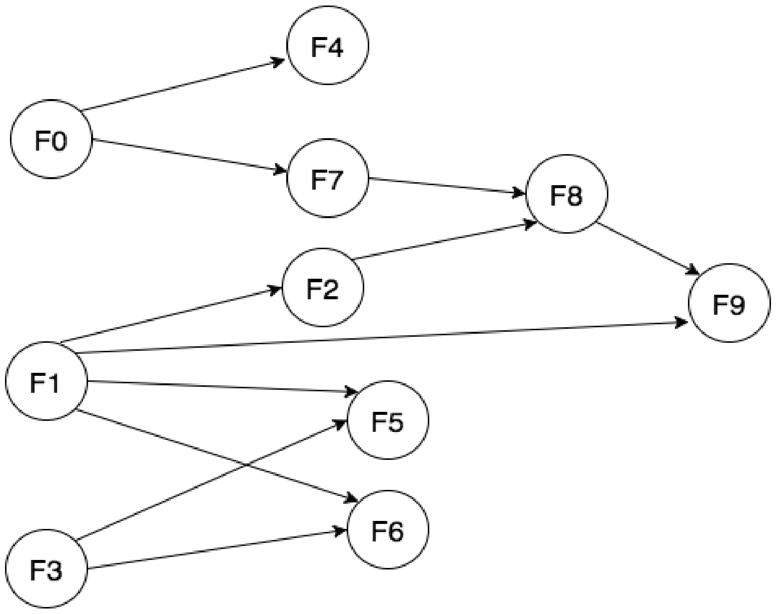
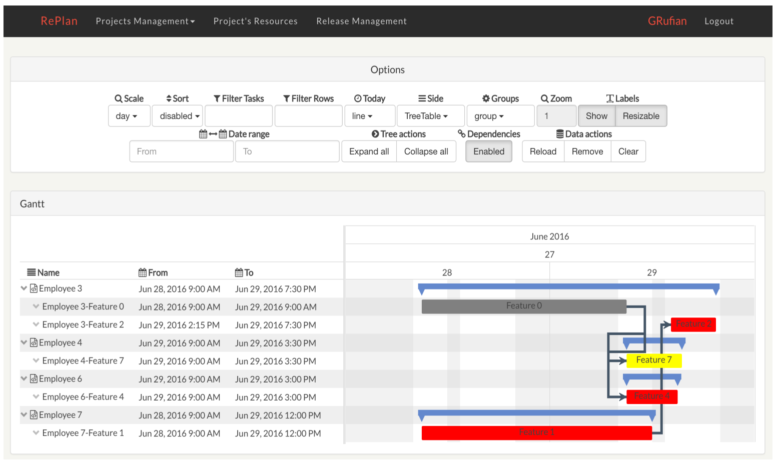


FIGURE 5.7: Graph of dependencies for Real Case Test 2

The final release plan for this test is shown in Figure 5.8



Scale: day; Sort: disabled; Today: line; Side: TreeTable; Groups: group; Zoom: 1; Labels: Show, Resizable; Date range: From, To; Dependencies: Enabled

Name	From	To
Employee 3	Jun 28, 2016 9:00 AM	Jun 29, 2016 7:30 PM
Employee 3-Feature 0	Jun 28, 2016 9:00 AM	Jun 29, 2016 9:00 AM
Employee 3-Feature 2	Jun 29, 2016 2:15 PM	Jun 29, 2016 7:30 PM
Employee 4	Jun 29, 2016 9:00 AM	Jun 29, 2016 3:30 PM
Employee 4-Feature 7	Jun 29, 2016 9:00 AM	Jun 29, 2016 3:30 PM
Employee 6	Jun 29, 2016 9:00 AM	Jun 29, 2016 3:00 PM
Employee 6-Feature 4	Jun 29, 2016 9:00 AM	Jun 29, 2016 3:00 PM
Employee 7	Jun 28, 2016 9:00 AM	Jun 29, 2016 12:00 PM
Employee 7-Feature 1	Jun 28, 2016 9:00 AM	Jun 29, 2016 12:00 PM

FIGURE 5.8: Release Planning for Real Case Test 2

5.2 Efficiency evaluation

5.2.1 Instance tests

In order to evaluate the efficiency of the RePlan engine, a set of instances $test_i$ have been extracted from [61]. The components of an instance consist in a set employees, features, skills, and the task precedence graph (TPG). Each of these components have several parameters that have to be translated into the entities working in the model.

Moreover, Alba et al. provide in [29] also an instance generator in JAVA capable of generating more instances randomly. In this case, a total of 36 different instances with different combinations of skills, features and resources have been created and tested, in order to study the influence of some parameters on the planning difficulty.

As the concepts of *employees* and *features* are clear in the context of this study, there are two slight difference at the moment of selecting the skills for each entity:

- The *employee skill* definition limits the amount of skills that an employee can have at a concrete moment. On the other hand, the amount of skills required to fulfill a task is random.

- The *general skill* definition limits the amount of skills that a feature can have at a concrete moment. On the other hand, the amount of each employee's skills is random.

To sum up, all the combinations that build this set of tests are expressed in Table 5.9:

	Employees Skills						General Skills					
	4-5 Skills			6-7 Skills			5 Skills			10 Skills		
Employees	5	10	15	5	10	15	5	10	15	5	10	15
Features												
10												
20												
30												

TABLE 5.10: Relation of generated instances

The next step is to execute each one of these tests into the RePlan engine, taking into account the *default configuration* settings of the tool:

- Only *one employee* can perform the totality of the task.
- The tasks with major priority are executed at first instance.
- The start time is the release's one. No replanning is performed.

Thus, the execution times (in seconds) of the different instances come as follows:

	Employees Skills						General Skills					
	4-5 Skills			6-7 Skills			5 Skills			10 Skills		
Employees	5	10	15	5	10	15	5	10	15	5	10	15
Features												
10	0.8	2.4	2.6	1.85	2.08	2.4	2.6	1.2	2	0.5	0.5	1.36
20	1.56	2.24	2.82	2.25	2.84	2.9	2.1	2.05	1.36	2.2	2.4	2.6
30	2.03	2.05	2.5	2.4	3.2	3.3	3.5	2.4	3	3.6	4	3.8

TABLE 5.11: Execution times of generated instances

5.3 Integration tests

In order to check that the different functionalities of the RePlan tool function properly, a set of integrated tests have been performed. Once each operation accomplishes the goals and runs inside the website tool, the following ones keep going in order to follow the life cycle of a normal project planning.

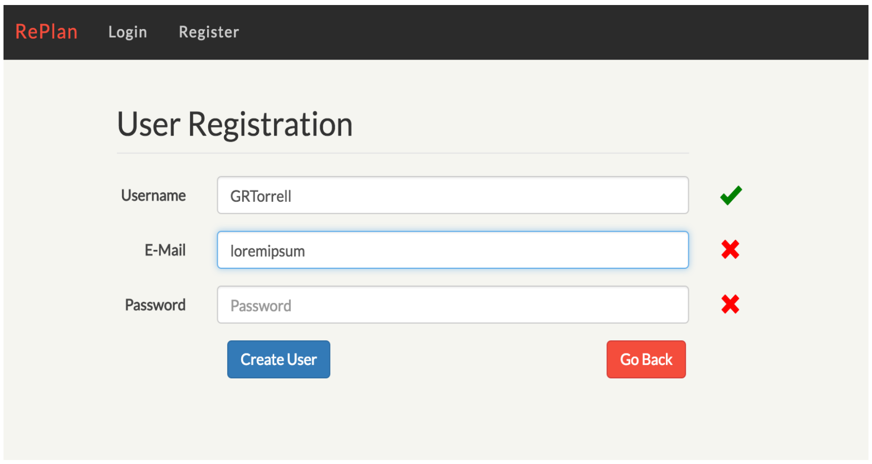
5.3.1 Operations inside the RePlan tool

The RePlan tool is based in a succession of basic operations that determine the lifecycle of a project inside the web application. Inside every operation, a set of tasks are performed and evaluated both in terms of completeness and efficiency (they fulfill the planned objective).

5.3.2 User and project registration

In order to guarantee security inside the tool, the first step that any user must perform at RePlan is registration. It is compulsory to guarantee that any unregistered user cannot have access to any other functionality inside the application, as she is not in possession of the adequate credentials (user ID).

When the user is registered into the tool, she introduces her personal data in an application view with different validators, as some fields are compulsory. In case all the fields are properly filled, a confirmation message appears. In any other case, the user will not be able to go forward.



Username	GRTorrell	✓
E-Mail	loremipsum	✗
Password	Password	✗

FIGURE 5.9: Test of invalid registration form at RePlan tool

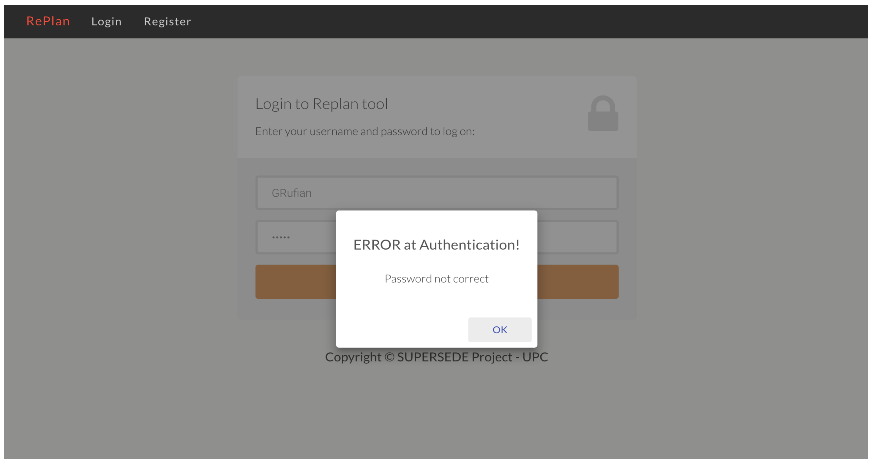
Once the user has been registered, she has the ability to create one or more projects. The process of creation is similar to the user's one: all fields are compulsory. The validation process is also immediate.

5.3.3 User login

In case the user is already registered and has a project created, she can *login* to the tool immediately.

If the validation process goes OK, the user can access immediately to the main dashboard of RePlan. As there is the case that a user can manage different projects, the *default* one will be the most recently created.

However, in the case there is any issue at the login, an *error message* must appear (see Figure 5.10).


RePlan
Login
Register
Login to Replan tool
Enter your username and password to log on:
GRufian
·····
OK
Copyright © SUPERSEDE Project - UPC

ERROR at Authentication!

Password not correct

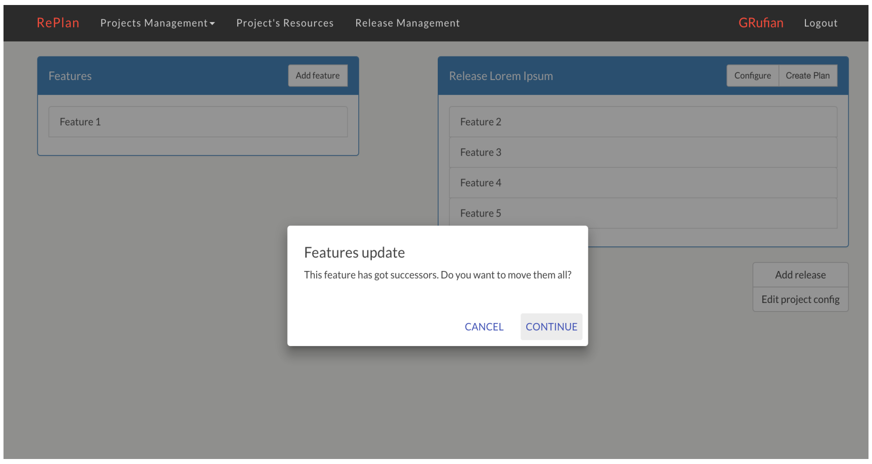
FIGURE 5.10: Test of incorrect login at RePlan tool

5.3.4 Main dashboard of RePlan

The dashboard of the RePlan tool is also the environment in which the user can manage her available releases. In this view, one of the most important points to validate is the proper execution of the features drag-and-drop selection. It would be illogical to insert a feature into a release if its ancestors are in subsequent releases or simply they are available. It is essential two consider two main cases:

- The feature has got *successors* and has to be deleted from a release. All the features related in succession dependency from the initial one must be also removed.

- The feature has got *predecessors* and is added to a releases. It would be illogical to insert a feature into a release if its ancestors are in subsequent releases or simply they are available. The tool has to indicate the user these actions are considered *illegal*.



Features update

This feature has got successors. Do you want to move them all?

FIGURE 5.11: Intent to remove a feature with successors in a release at RePlan's main screen

Apart from this special validation, the rest of the validation respond to a proper functioning of the different actions in the environment. It deserves a special mention the *creation* of a scheduled plan, as it is the most complex operation of all the project. In case there is any error, an appropriate alert message indicates it to the user and *rolls back* the changes.

5.3.5 Project features management

The creation and edition of a feature has the same structure (in terms of validation) as the registration or project creation forms.

In this case, the validation is only prevailing in some essential attributes. A feature might not have any dependency nor specific skills. All features with no skills are assigned to a skill called *General* by default.

5.3.6 Project resources management

In this view of the tool, the user is able to add or delete as many resources (i.e. employees) with different features. It is essential to validate that the employees skills (i.e. capacities) can only be added from the corresponding modal view.

Moreover, it is also checked that the user name and availability must be fulfilled. In the case of skills, as it has been told in aforementioned sections, there is no obligation in adding anyone. All employees with no skills are assigned to a skill called *General* by default.

5.3.7 Scheduled planning

Once all the parameters have been set-up (i.e. releases, resources and features), the scheduled planning can be generated. As in Section 5.1 a great amount of planning tests have been performed, the goal is to check some of the options of the configuration panel.

5.4 Discussion

This section will deal with both the discussion of the RePlan tool's efficiency in the different cases proposed in the previous sections. Moreover, an analysis of the integration tests is also provided.

In order to perform the evaluation of the scheduling plans library (implemented in jMetal), we have decided to set-up the metaheuristic algorithm with 150 iterations and a population of 150 solutions. These values are modifiable up and down, in order to make the metaheuristic perform the evaluation, selection and mutation of the population more or less times.

This total amount of iterations and the size of the population have been selected due to the good approximations they provide in other studies like [39] and [18]. Although in many occasions the metaheuristic will not provide the most optimal solution to a problem, these values (or even higher ones) could give very close approximations to the best solution possible. Moreover, as the metaheuristic keeps iterating, it might arrive to a moment in the execution in which all the solutions *are identical*. This means that the process has come up to a point at which it cannot create new solutions and it is performing *unnecessary operations*.

The different experiments done with the provided instances demonstrate that, even in the worst cases, the average execution times are not higher than 4 seconds. This time can be considered as acceptable, if we take into account that the metaheuristic inner processing (i.e. selection and mutation processing) takes out a good part of the total.

On the other hand, all the integration tests performed show that the tool can perform all the required operations successfully. Moreover, taking into account the different *stress tests* performed, the tool can recover from different sorts of internal errors and rollback the state of the tool to avoid losing the already done process.

Chapter 6

Conclusions

The RePlan tool covers two different well-known topics in the literature. On the right hand, the decision of which features can appear in one (or more) concrete releases. On the other hand, this decision is combined with a resources planning efficient algorithm.

The result is a high-modular engine and essentially multi-objective. In any moment more objective functions can be added to the problem, in order to modify and diversify the desired results.

Moreover, the design and implementation of the tool extends from this base. This guarantees good pairing from the engine to a great amount of solutions and projects.

On the other hand, the RePlan tool provides to the decision-makers a new focus on the project management environment. The combination between the front-end technologies and the efficiency of the back-end part guarantee a complete user experience. Besides, this solution offers different functionalities from the other options that exist nowadays in the market.

As a future work, the great flexibility of the metaheuristic algorithms guarantee many uses and options in the RePlan problem. Among other options the *minimization of project risks* can be added, or even sum multiple criteria from different project stakeholders.

About the front-end part, it is always necessary to adapt the stylesheets and underlying technologies (Angular and JavaScript) to new functionalities (eg. drag and drop, among others).

Bibliography

[1] App Stats. *App Usage Statistics: 2015 Roundup*. 2015. URL: `http://www.businessofapps.com/app-usage-statistics-2015/` (visited on 06/16/2016).

[2] Björn Regnell and Sjaak Brinkkemper. "Market-driven requirements engineering for software products". In: (2005), pp. 287–308.

[3] Pär Carlshamre and Björn Regnell. "Requirements lifecycle management and release planning in market-driven requirements engineering processes". In: (2000), pp. 961–965.

[4] Juan J Durillo et al. "A study of the multi-objective next release problem". In: *1st International Symposium on Search Based Software Engineering*. IEEE. 2009, pp. 49–58.

[5] Zengqiang Jiang, Le Zuo, and E Mingcheng. "Study on multi-objective flexible job-shop scheduling problem considering energy consumption". In: *Journal of Industrial Engineering and Management* 7.3 (2014), p. 589.

[6] Günther Ruhe. *Product release planning: methods, tools and applications*. CRC Press, 2010.

[7] Des Greer and Günther Ruhe. "Software release planning: an evolutionary and iterative approach". In: *Information and Software Technology* 46.4 (2004), pp. 243–253.

[8] Expert Decisions. *Release Planner*. 2015. URL: `http://rp2.releaseplanner.com` (visited on 06/16/2016).

[9] UPC. *SUPERSEDE*. 2014. URL: `https://www.supersede.eu` (visited on 05/17/2016).

[10] Martin Fowler and Jim Highsmith. “The agile manifesto”. In: *Software Development* 9.8 (2001), pp. 28–35.

[11] Phillipe Kruchten. *From Waterfall to Iterative Lifecycle-a tough transition for project managers*. 2000.

[12] James E Kelley. “The critical-path method: Resources planning and scheduling”. In: *Industrial scheduling* 13 (1963), pp. 347–365.

[13] Mark Gammon and Viktoriya Oliynyk. “Trello”. In: (2015).

[14] Mikael Svahnberg et al. “A systematic review on strategic release planning models”. In: *Information and software technology* 52.3 (2010), pp. 237–248.

[15] Axel Van Lamsweerde. “Requirements engineering in the year 00: a research perspective”. In: *Proceedings of the 22nd international conference on Software engineering*. ACM. 2000, pp. 5–19.

[16] Kalyanmoy Deb et al. “A fast and elitist multiobjective genetic algorithm: NSGA-II”. In: *Evolutionary Computation, IEEE Transactions on* 6.2 (2002), pp. 182–197.

[17] Hui Li and Qingfu Zhang. “Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II”. In: *Evolutionary Computation, IEEE Transactions on* 13.2 (2009), pp. 284–302.

[18] Yuanyuan Zhang, Mark Harman, and S Afshin Mansouri. “The multi-objective next release problem”. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM. 2007, pp. 1129–1137.

[19] Steven Haines. *Open Source Java Projects*. 2014. URL: `http://www.javaworld.com/blog/open-source-java-projects/` (visited on 05/22/2016).

[20] Juan J Durillo and Antonio J Nebro. “jMetal: A Java framework for multi-objective optimization”. In: *Advances in Engineering Software* 42.10 (2011), pp. 760–771.

[21] Pivotal. *Spring Framework - Introduction*. 2014. URL: `http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html` (visited on 05/22/2016).

[22] Oliver White. *Top 4 Java Web Frameworks Revealed: Real Life Usage Data of Spring MVC, Vaadin, GWT and JSF*. 2015. URL: `http://zeroturnaround.com/rebellabs/top-4-java-web-frameworks-revealed-real-life-usage-data-of-spring-mvc-vaadin-gwt-and-jsf/` (visited on 05/22/2016).

[23] Anthony J. Bagnall, Victor J. Rayward-Smith, and Ian M Whittley. "The next release problem". In: *Information and software technology* 43.14 (2001), pp. 883–890.

[24] Arezou Mohammadi. *Scheduling Algorithms for Real-Time Systems*. Tech. rep. 2005.

[25] Chen Li et al. "An integrated approach for requirement selection and scheduling in software release planning". In: *Requirements engineering* 15.4 (2010), pp. 375–396.

[26] Michail G Lagoudakis. "The 0–1 Knapsack Problem". In: *reason* 2 (), u2U0.

[27] Aristide Mingozzi et al. "An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation". In: *Management Science* 44.5 (1998), pp. 714–729.

[28] Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-constrained project scheduling: models, algorithms, extensions and applications*. John Wiley & Sons, 2013.

[29] Enrique Alba and J Francisco Chicano. "Software project management with GAs". In: *Information Sciences* 177.11 (2007), pp. 2380–2401.

[30] Erik Demeulemeester and Willy Herroelen. "A branch-and-bound procedure for the multiple resource-constrained project scheduling problem". In: *Management science* 38.12 (1992), pp. 1803–1818.

[31] Toni Frankola, Marin Golub, and Domagoj Jakobovic. "Evolutionary algorithms for the resource constrained scheduling problem". In: *30th International Conference on Information Technology Interfaces, ITI 2008*. 2008.

[32] Seth Ladd et al. *Expert Spring MVC and Web Flow*. Vol. 1. Springer, 2006.

[33] Leonard Richardson and Sam Ruby. *RESTful web services*. " O'Reilly Media, Inc.", 2008.

[34] Roy Thomas Fielding. "Architectural styles and the design of network-based software architectures". PhD thesis. University of California, Irvine, 2000.

[35] Roy T Fielding and Richard N Taylor. "Principled design of the modern Web architecture". In: *ACM Transactions on Internet Technology (TOIT)* 2.2 (2002), pp. 115–150.

[36] Arnaud Cogoluègnes. "HATEOAS paging with Spring MVC and Spring Data JPA". In: (2012).

[37] Roy Fielding et al. *Hypertext transfer protocol–HTTP/1.1*. 1999.

[38] Robert C Martin. *SRP: The Single Responsibility Principle*. 1996.

[39] Muhammad Rezaul Karim and Guenther Ruhe. "Bi-objective genetic search for release planning in support of themes". In: *Search-Based Software Engineering*. Springer, 2014, pp. 123–137.

[40] Qingfu Zhang and Hui Li. "MOEA/D: A multiobjective evolutionary algorithm based on decomposition". In: *Evolutionary Computation, IEEE Transactions on* 11.6 (2007), pp. 712–731.

[41] Joshua Knowles and David Corne. "The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation". In: *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*. Vol. 1. IEEE. 1999.

[42] Bruce Momjian. *PostgreSQL: introduction and concepts*. Vol. 192. Addison-Wesley New York, 2001.

[43] Tim Downey. *Web development with java: using hibernate, JSPs and Servlets*. Springer Science & Business Media, 2008.

[44] Tom Boyle. "Design principles for authoring dynamic, reusable learning objects". In: *Australian Journal of Educational Technology* 19.1 (2003), pp. 46–58.

[45] Mark Pilgrim. *HTML5: up and running*. " O'Reilly Media, Inc.", 2010.

[46] David Cochran. *Twitter Bootstrap Web Development How-To*. Packt Publishing Ltd, 2012.

[47] Brad Green and Shyam Seshadri. *AngularJS*. " O'Reilly Media, Inc.", 2013.

[48] Marcel Juenemann. *Angular use statistics*. 2016. URL: `http://libscore.com/?#angular` (visited on 06/11/2016).

[49] Elizabeth J O'Neil. "Object/relational mapping 2008: hibernate and the entity data model (edm)". In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM. 2008, pp. 1351–1356.

[50] Masoud Kalali and Bhakti Mehta. *Developing RESTful services with JAX-RS 2.0, WebSockets, and JSON*. Packt Publishing Ltd, 2013.

[51] Elie Feirouz et al. *Method and system for storing a web browser application session cookie from another client application program*. US Patent App. 11/167,787. 2005.

[52] Marcel Juenemann. *Angular Drag-and-Drop lists*. 2016. URL: `http://marceljuenemann.github.io/angular-drag-and-drop-lists` (visited on 06/01/2016).

[53] Google Inc. *Packaged Angular Messages*. 2015. URL: `https://github.com/angular/bower-angular-messages` (visited on 06/05/2016).

[54] Ashish Shukla. *Routing in MVC*. 2013. URL: `http://www.codeproject.com/Tips/573454/Routing-in-MVC` (visited on 08/06/2016).

[55] Dan Wahlin. "AngularJS in 60 Minutes". In: *Wahlin Consulting* 2014 (2013).

[56] Creately. *5 Reasons to user Gantt charts*. 2012. URL: `http://creately.com/blog/diagrams/5-reasons-to-use-gantt-charts/` (visited on 06/08/2016).

[57] Marco Schweighauser and Rémi Alvergnat. *Angular-Gantt plug-in for AngularJS*. 2015. URL: `http://www.angular-gantt.com` (visited on 05/20/2016).

[58] John Ferguson Smart et al. "An introduction to Maven 2". In: *Java-World Magazine. Available at: http://www. javaworld. com/javaworld/jw-12-2005/jw-1205-maven. html* (2005).

[59] Aleksa Vukotic and James Goodwill. *Apache Tomcat 7*. Springer, 2011.

[60] Neil Middleton, Richard Schneeman, et al. *Heroku: Up and Running*. " O'Reilly Media, Inc.", 2013.

[61] Francisco Chicano. *An Instance Generator for the Project Scheduling Problem*. 2005. URL: `http://tracer.lcc.uma.es/problems/psp/generator.html` (visited on 03/23/2016).

Primary studies for the academic state of the art

[M1] Mark Przepiora, Reza Karimpour, and Guenther Ruhe. *A hybrid release planning method and its empirical justification*. ACM, 2012.

[M2] Gert van Valkenhoef et al. *Quantitative release planning in extreme programming*. 2011.

[M3] Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia. *Multi-sprint planning and smooth replanning: An optimization model*. 2013.

[M4] Jifeng Xuan et al. "Solving the large scale next release problem with a backbone-based multilevel algorithm". In: *Software Engineering, IEEE Transactions on* 38.5 (2012), pp. 1195–1212.

[M5] Gabriele Zorn-Pauli et al. "Analyzing an industrial strategic release planning process–a case study at Roche diagnostics". In: *Requirements Engineering: Foundation for Software Quality*. Springer, 2013, pp. 269–284.

[M6] Ville T Heikkilä et al. *Continuous release planning in a large-scale scrum development organization at Ericsson*. Springer, 2013.

[M7] Jason McZara et al. "Software requirements prioritization and selection using linguistic tools and constraint solvers—a controlled experiment". In: *Empirical Software Engineering* 20.6 (2015), pp. 1721–1761.

[M8] Michael Felderer et al. "Industrial evaluation of the impact of quality-driven release planning". In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM. 2014, p. 62.

[M9] Nishant Agarwal, Reza Karimpour, and Guenther Ruhe. “Theme-based product release planning: An analytical approach”. In: *System Sciences (HICSS), 2014 47th Hawaii International Conference on*. IEEE. 2014, pp. 4739–4748.

[M10] Stefan Gueorguiev, Mark Harman, and Giuliano Antoniol. “Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering”. In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM. 2009, pp. 1673–1680.

[M11] Ahmed Al-Emran et al. “Studying the impact of uncertainty in operational release planning–An integrated method and its initial evaluation”. In: *Information and Software Technology* 52.4 (2010), pp. 446–461.

[M12] Ville Heikkilä et al. “Rigorous support for flexible planning of product releases-a stakeholder-centric approach and its initial evaluation”. In: *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*. IEEE. 2010, pp. 1–10.

[M13] Parmeet Kaur. “Reinforcement learning based approach for adaptive release planning in an agile environment”. In: *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*. IEEE. 2010, pp. 1–4.

[M14] Ákos Szőke. “Conceptual scheduling model and optimized release scheduling for agile environments”. In: *Information and software technology* 53.6 (2011), pp. 574–591.

[M15] Francisco Luna et al. “The software project scheduling problem: A scalability analysis of multi-objective metaheuristics”. In: *Applied Soft Computing* 15 (2014), pp. 136–148.

[M16] José M Chaves-González and Miguel A Pérez-Toledano. “Differential evolution with Pareto tournament for the multi-objective next release problem”. In: *Applied Mathematics and Computation* 252 (2015), pp. 1–13.

[M17] Chen Li et al. "An integrated approach for requirement selection and scheduling in software release planning". In: *Requirements engineering* 15.4 (2010), pp. 375–396.

[M18] Muhammad Rezaul Karim and Guenther Ruhe. "Bi-objective genetic search for release planning in support of themes". In: *Search-Based Software Engineering*. Springer, 2014, pp. 123–137.

[M19] Antonio Mauricio Pitangueira et al. "Risk-Aware Multi-stakeholder Next Release Planning Using Multi-objective Optimization". In: *Requirements Engineering: Foundation for Software Quality*. Springer, 2016, pp. 3–18.

[M20] Fatma Başak Aydemir et al. "Next Release Tool". In: ().

[M21] Günther Ruhe. *Product release planning: methods, tools and applications*. CRC Press, 2010.

[M22] Juan J Durillo et al. "A study of the bi-objective next release problem". In: *Empirical Software Engineering* 16.1 (2011), pp. 29–60.

Appendix A

Replan Tool API

Introduction

RePlan Tool API

Implement your planning tool using the RePlan Tool API

Version:	1.0.0
Host:	replanupc.herokuapp.com
Base Path:	/v1
Scheme:	https

Generated on: Mon Jun 20 2016 23:22:11 GMT+0200 (CEST)

Table of contents

2.14. get /skills/search

2.15. get /releases

2.16. post /releases

2.17. delete /releases

2.18. get /releases/search

2.19. put /releases/updateRequirements

2.20. get /plans

2.21. delete /plans

2.22. get /plans/generatePlan

1. Definitions

1.1. UserLogin

Name	Type	Description	Required
login_id	string	Unique session identifier representing the access of a user into the RePlan tool.	Yes
username	string		No

1.2. UserInfoDTO

Name	Type	Description	Required
username	string		Yes
projectId	integer		Yes

1.3. NewUserDTO

Name	Type	Description	Required
username	string		Yes
email	string		Yes
password	string		Yes

1.4. RequirementDTO

Name	Type	Description	Required
id	integer		Yes
internalId	integer		Yes
name	string		Yes
effort	integer		Yes
criteria	integer		Yes
maxEmployees	integer		Yes
deadline	string		Yes
successors	array	See **RequirementDTO** in the **Definitions** section.	Yes
		See **RequirementDTO**	

predecessors	array	in the **Definitions** section.	Yes
skills	array	See **SkillDTO** in the **Definitions** section.	Yes
state	string		Yes
isPlanned	boolean		Yes

1.5. SkillDTO

Name	Type	Description	Required
id	integer		No
name	string		Yes
abbr	string		Yes
description	string		Yes

1.6. EmployeeDTO

Name	Type	Description	Required
id	integer		No
name	string		Yes
effort	integer		Yes
skills	array	See **SkillDTO** in the **Definitions** section.	Yes

1.7. ReleaseDTO

Name	Type	Description	Required
id	integer		No
name	string		Yes
description	string		No
initialDate	string		Yes
deadline	string		Yes
hasPlans	boolean		No
resources	array	See **EmployeeDTO** in the **Definitions** section.	No
requirements	array	See **RequirementDTO** in the **Definitions** section.	No

1.8. PlanDTO

Name	Type	Description	Required
id	integer		No
projectId	integer		Yes
name	string		Yes
releaseDate	string		Yes
objects	string		Yes

1.9. Error

Name	Type	Description	Required
code	integer		No
message	string		No
fields	string		No

2. Paths

2.2 get /users

Summary	User Information
Description	The User endpoint returns information about the user. The response includes the c project at which she is working and the list of her available projects.
Operation Id	GetUserInfo
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collec Forma
userLogin	body	Login information of the user.	Yes			

Name	Type	Required
login_id	string	Yes
username	string	No

Responses

code	description
200	The information of the user. See **UserInfoDTO** in the **Definitions** sectio
204	No content
401	Unauthorized
501	Internal server error

Name	Type	Requ
username	string	Yes
projectId	integer	Yes

2.2 post /users

Summary	Save user information
Description	The User endpoint saves the information from the new registered user.
Operation Id	CreateUser
Produces	application/json
Consumes	application/json

	Name	In	Description	Required	Type	Format	Collection Format
Parameters	user	body	New user to be added **Name** / **Type** / **Required** username / string / Yes email / string / Yes password / string / Yes	Yes			

Responses

code	description
201	User Created" See **UserInfoDTO** in the **Definitions** section
401	Unauthorized
501	Internal server error

Name	Type	Requir
username	string	Yes
projectId	integer	Yes

2.3 get /users/changeProject

Summary	Change user project
Description	The User endpoint changes the current project to another one specified by parame
Operation Id	changeUserProject
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Default
projectId	query	ID of the project to work on right now.	Yes	integer	int64		

Responses

code	description
200	The information of the user. See **UserInfoDTO** in the **Definitions** section.

Name	Type	Required
username	string	Yes
projectId	integer	Yes

code	description
default	Unexpected error See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No

2.4 get /requirements

Summary	Search Requirement created by Requirement ID.
Description	The Requirements endpoint retrieve the Requirement identified by requirementID.
Operation Id	searchRequirementById
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Def
requirementId	query	ID of the requirement.	Yes	integer	int64		

Responses

code	description
200	Successful response See **RequirementDTO** in the **Definitions** secti

Name	Type	Required
id	integer	Yes
internalId	integer	Yes
name	string	Yes
effort	integer	Yes
criteria	integer	Yes
maxEmployees	integer	Yes
deadline	string	Yes
successors	array	Yes
predecessors	array	Yes
skills	array	Yes
state	string	Yes
isPlanned	boolean	Yes

code	description
default	Unexpected error

See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No

2.4 post /requirements

Summary	Save requirement information
Description	The Requirements endpoint saves the information from the new created requireme
Operation Id	saveRequirement
Produces	application/json
Consumes	application/json

Parameters

Name	In	Description	Required	Type	Forma
requirement	body	New requirement to be added to the Project	Yes		

Name	Type	Required
id	integer	Yes
internalId	integer	Yes
name	string	Yes
effort	integer	Yes
criteria	integer	Yes
maxEmployees	integer	Yes
deadline	string	Yes
successors	array	Yes
predecessors	array	Yes
skills	array	Yes
state	string	Yes
isPlanned	boolean	Yes

Responses

code	description
201	Requirement Created"

See **RequirementDTO** in the **Definition**

Name	Type
id	integer

internalId	integer
name	string
effort	integer
criteria	integer
maxEmployees	integer
deadline	string
successors	array
predecessors	array
skills	array
state	string
isPlanned	boolean

401	Unauthorized
501	Internal server error

2.4 delete /requirements

Summary	Delete requirement information
Description	The Requirements endpoint deletes the requirement identified by ID.
Operation Id	deleteRequirement
Produces	application/json
Consumes	application/json

Parameters

Name	In	Description	Required	Type	Format	Collect Format
deletedRequirementId	query	Requirement to be deleted.	Yes	integer	int64	

Responses

code	description
200	"Requirement Deleted"
401	Unauthorized
501	Internal server error

2.5 get /requirements/search

Summary	Search Requirements created inside a Project.
Description	The Requirements endpoint retrieves all the requirements created in a project.
Operation	searchRequirementsByProjectId

Id	
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Default
projectId	query	ID of the project.	Yes	integer	int64		

Responses

code	description
200	Successful response

Schema type array

See **RequirementDTO** in the **Definitions** sectior

Name	Type	Required
id	integer	Yes
internalId	integer	Yes
name	string	Yes
effort	integer	Yes
criteria	integer	Yes
maxEmployees	integer	Yes
deadline	string	Yes
successors	array	Yes
predecessors	array	Yes
skills	array	Yes
state	string	Yes
isPlanned	boolean	Yes

code	description
default	Unexpected error

See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No

2.6 get /requirements/searchAvailable

Summary Search available Requirements created inside a Project.

Description The Requirements endpoint retrieves all the requirements created in a project that

	been included** in any release.
Operation Id	searchAvailableRequirementsByProjectId
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Default
projectId	query	ID of the project.	Yes	integer	int64		

Responses

code	description
200	Successful response

Schema type array

See **RequirementDTO** in the **Definitions** sectior

Name	Type	Required
id	integer	Yes
internalId	integer	Yes
name	string	Yes
effort	integer	Yes
criteria	integer	Yes
maxEmployees	integer	Yes
deadline	string	Yes
successors	array	Yes
predecessors	array	Yes
skills	array	Yes
state	string	Yes
isPlanned	boolean	Yes

code	description
default	Unexpected error

See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No

2.7 get /employees

Summary	Search Employee created by Employee ID.
Description	The Employee endpoint retrieve the Employee identified by employeeID.
Operation Id	searchEmployeeById
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Defau
employeeId	query	ID of the employee.	Yes	integer	int64		

Responses

code	description
200	Successful response

See **EmployeeDTO** in the **Definitions** section.

Name	Type	Required
id	integer	No
name	string	Yes
effort	integer	Yes
skills	array	Yes

code	description
default	Unexpected error

See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No

2.7 post /employees

Summary	Save employee information
Description	The Employees endpoint saves the information from the new created employee.
Operation Id	saveEmployee
Produces	application/json
Consumes	application/json

Parameters

Name	In	Description	Required	Type	Format	Collectio Format
employee	body	New employee to be	Yes			

added to the Project

Name	**Type**	**Required**
id	integer	No
name	string	Yes
effort	integer	Yes
skills	array	Yes

Responses

code	**description**
201	Employee Created"
	See **EmployeeDTO** in the **Definitions** sectio

Name	**Type**	**Requir**
id	integer	No
name	string	Yes
effort	integer	Yes
skills	array	Yes

401	Unauthorized
501	Internal server error

2.7 delete /employees

Summary	Delete employee information
Description	The Employees endpoint deletes the employee identified by ID.
Operation Id	deleteEmployee
Produces	application/json
Consumes	application/json

Parameters

Name	**In**	**Description**	**Required**	**Type**	**Format**	**Collection Format**
deletedEmployeeId	query	Employee to be deleted.	Yes	integer	int64	

Responses

code	**description**
200	"Employee Deleted"
401	Unauthorized
501	Internal server error

2.8 post /skills

Summary	Save skill information
Description	The Skill endpoint saves the information from the new created skill.
Operation Id	saveSkill
Produces	application/json
Consumes	application/json

Parameters

Name	In	Description	Required	Type	Format	Collecti Format
skill	body	New skill to be added	Yes			

Name	Type	Required
id	integer	No
name	string	Yes
abbr	string	Yes
description	string	Yes

Responses

code	description
201	Skill Created"

See **SkillDTO** in the **Definitions** section.

Name	Type	Requ
id	integer	No
name	string	Yes
abbr	string	Yes
description	string	Yes

code	description
401	Unauthorized
501	Internal server error

2.8 delete /skills

Summary	Delete skill information
Description	The Skill endpoint deletes the skill identified by ID.
Operation Id	deleteSkill
Produces	application/json
Consumes	application/json

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Defa
deletedSkillId	query	Skill to be	Yes	integer	int64		

	deleted.

Responses

code	description
200	"Skill Deleted"
401	Unauthorized
501	Internal server error

2.9 get /skills/search

Summary	Search Skills created inside a Project.
Description	The Skills endpoint retrieves all the skills created in a project.
Operation Id	searchSkillsByProjectId
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Default
projectId	query	ID of the project.	Yes	integer	int64		

Responses

code	description
200	Successful response

Schema type array

See **SkillDTO** in the **Definitions** section.

Name	Type	Required
id	integer	No
name	string	Yes
abbr	string	Yes
description	string	Yes

code	description
default	Unexpected error

See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No

2.10 get /releases

Summary	Search Release created by Release ID.
Description	The Release endpoint retrieve the Release identified by releaseID.
Operation Id	searchReleaseById
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Default
releaseId	query	ID of the release.	Yes	integer	int64		

Responses

code	description
200	Successful response

See **ReleaseDTO** in the **Definitions** section.

Name	Type	Required
id	integer	No
name	string	Yes
description	string	No
initialDate	string	Yes
deadline	string	Yes
hasPlans	boolean	No
resources	array	No
requirements	array	No

code	description
default	Unexpected error

See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No

2.10 post /releases

Summary	Save release information
Description	The Releases endpoint saves the information from the new created release.
Operation	saveRelease

Id

Produces	application/json
Consumes	application/json

Parameters

Name	In	Description	Required	Type	Format	Col For
release	body	New release to be added to the Project	Yes			

Name	Type	Required
id	integer	No
name	string	Yes
description	string	No
initialDate	string	Yes
deadline	string	Yes
hasPlans	boolean	No
resources	array	No
requirements	array	No

Responses

code	description
201	Release Created"

See **ReleaseDTO** in the **Definitions** sectic

Name	Type	R
id	integer	N
name	string	Y
description	string	N
initialDate	string	Y
deadline	string	Y
hasPlans	boolean	N
resources	array	N
requirements	array	N

code	description
401	Unauthorized
501	Internal server error

2.10 delete /releases

Summary	Delete release information
Description	The Releases endpoint deletes the release identified by ID.
Operation	deleteRelease

Id

Produces application/json

Consumes application/json

Parameters

Name	In	Description	Required	Type	Format	Collection Format	D
deletedReleaseId	query	Release to be deleted.	Yes	integer	int64		

Responses

code	description
200	"Release Deleted"
401	Unauthorized
501	Internal server error

`2.11 get  /releases/search`

Summary Search Releases created inside a Project.

Description The Releases endpoint retrieves all the releases created in a project.

Operation Id searchReleasesByProjectId

Produces

Consumes

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Default
projectId	query	ID of the project.	Yes	integer	int64		

Responses

code	description
200	Successful response

Schema type array

See **ReleaseDTO** in the **Definitions** section.

Name	Type	Required
id	integer	No
name	string	Yes
description	string	No
initialDate	string	Yes
deadline	string	Yes
hasPlans	boolean	No

resources	array	No
requirements	array	No

code	description
default	Unexpected error See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No

2.12 put /releases/updateRequirements

Summary	Updates uniquely requirements from a Release.
Description	The release identified by releaseID updates uniquely the requirements passed by p
Operation Id	updateReleaseRequirements
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Forn
releaseId	query	ID of the release.	Yes	integer	int64
requirements	body	ID of the release.	Yes		

Name	Type	Required
id	integer	No
name	string	Yes
description	string	No
initialDate	string	Yes
deadline	string	Yes
hasPlans	boolean	No
resources	array	No
requirements	array	No

Responses

code	description
200	"Release updated"
default	Unexpected error See **Error** in the **Definitions** section.

Name	Type
code	integer

message	string
fields	string

2.13 get /plans

Summary	Search Plans created for a release
Description	The Plans endpoint retrieves all the plans created for a release.
Operation Id	searchPlansByReleaseId
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Default
releaseId	query	ID of the release.	Yes	integer	int64		

Responses

code	description
200	Successful response
default	Unexpected error

200 — Successful response

Schema type array

See **PlanDTO** in the **Definitions** section.

Name	Type	Required
id	integer	No
projectId	integer	Yes
name	string	Yes
releaseDate	string	Yes
objects	string	Yes

default — Unexpected error

See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No

2.13 delete /plans

Summary	Delete scheduled plans from a release.
Description	The Plans endpoint deletes all plans created for a release identified by release ID.
Operation Id	deletePlans
Produces	application/json
Consumes	application/json

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Default
releaseId	query	Release whose plans are deleted.	Yes	integer	int64		

Responses

code	description
200	"Plans Deleted"
401	Unauthorized
501	Internal server error

2.14 get /plans/generatePlan

Summary	Generate plans for a Release.
Description	The Plans endpoint creates scheduled plans for an identified Release.
Operation Id	GeneratePlans
Produces	
Consumes	

Parameters

Name	In	Description	Required	Type	Format	Collection Format	Default
releaseId	query	ID of the release.	Yes	integer	int64		

Responses

code	description
200	"Plans Created"
default	Unexpected error

See **Error** in the **Definitions** section.

Name	Type	Required
code	integer	No
message	string	No
fields	string	No