Universitat Oberta de Catalunya

uoc.edu

Automated Similarity Detection: Identifying Duplicated Requirements

Author	**Quim Motger de la Encarnación**
Studies	Master in Informatics Engineering
Department	Artificial Intelligence Department
Thesis Supervisor	**Cristina Palomares Bonache**
Chief of Department	**Carles Ventura Royo**

December 2019

Master Thesis Report

Thesis title:	*Automated Similarity Detection: Identifying Duplicated Requirements*
Author:	*Quim Motger de la Encarnación*
Advisor:	*Cristina Palomares Bonache*
PRA:	*Carles Ventura Royo*
Submission date:	12/2019
Studies:	*Master in Informatics Engineering*
Master Thesis Department:	*Artificial Intelligence (subject: IAA)*
Language:	*English*
Keywords:	*requirements engineering, similarity detection, duplicated requirements*

Abstract (English):

Machine-Learning (ML) and Natural-Language-Processing (NLP) are two of the most known areas of Artificial Intelligence (AI). ML is a general-purpose technology which uses data to learn real-world knowledge and to improve the reliability of a specific action - typically to extract autonomous predictions about partial data observations. On the other hand, NLP applies to the task of developing representations of features of natural language based on its textual information.

One area of application of NLP and ML is the *Requirements Engineering* (RE) field. RE is the set of processes of *Software Engineering* (SE) focused on the management of a set of requirements that describes a system. Between the challenges of RE, it is highlighted the detection of duplicated requirements. If ignored, these duplicities may lead to redundancy in the textual information of a project and therefore this may lead to the duplicity of tasks. Moreover, the automation of this process and the standardized usage of specific, accurate tools are still at a state-of-the-art stage.

This master thesis is a state-of-the-art analysis to apply automated requirements similarity detection, using AI techniques, for the detection of duplicates between project requirements. Based on a literature review, this thesis must be a practical evaluation and a development proposal of duplicate detection in SE project requirements.

This work is developed within the OpenReq project, an *EU-Horizon-2020 project* whose goal is *"to build an intelligent decision system for community-driven RE"*. This collaboration allows the usage of real requirements data to evaluate the algorithms developed in this project.

Abstract (Catalan):

Machine-Learning (ML) i *Natural-Language-Processing* (NLP) són dues de les principals àrees del camp de la Intel·ligència Artificial (IA). ML és una tecnologia de propòsit genèric que utilitza l'anàlisi de dades per extreure coneixement del món real i millorar la fiabilitat d'un procés, generalment aplicable a extreure prediccions autònomes basades en observacions parcials de dades. Per altra banda, NLP

consisteix en el desenvolupament de representacions computacionals i el processament de característiques del llenguatge natural.

Una de les seves principals àrees d'aplicació és l'Enginyeria de Requisits (RE), que consisteix en el conjunt de processos de l'Enginyeria Software (SE) relacionats amb la gestió del conjunt de requisits d'un sistema. Entre els reptes de la RE destaca la detecció de requisits duplicats. Aquestes duplicitats, si no es gestionen, poden comportar una redundància en la informació textual d'un projecte, i en conseqüència, la duplicitat de tasques. Tot i així, l'automatització d'aquest procés i l'ús d'eines estandaritzades es troba encara en fase de desenvolupament.

L'objectiu d'aquesta tesi és analitzar l'estat de l'art de la detecció automàtica de similitud entre requisits d'un projecte utilitzant tècniques d'AI. D'acord amb aquesta recerca, es planteja una avaluació pràctica i una proposta de desenvolupament d'un sistema per la detecció de duplicats entre requisits d'un projecte de SE.

Aquesta tesi s'enmarca en el projecte OpenReq del programa *EU-Horizon-2020*. L'objectiu d'aquest és *"implementar un sistema de decisió intel·ligent basat en la comunitat de RE"*. Aquesta col·laboració permet l'ús de dades reals de requisits per avaluar els algoritmes desenvolupats en aquest projecte.

Index

Acknowledgements

This project has been developed in cooperation with the Software and Service Engineering Group (GESSI) at Universitat Politècnica de Catalunya (UPC). The work and results presented in this paper have been conducted within the scope of the Horizon 2020 project OpenReq, which is supported by the European Union under the Grant Nr. 732463.

Particularly, I would like to thank the Qt Group, a global software company involved as a partner in the OpenReq project, for allowing the access and the use of the data used in this dissertation.

Special thanks to Cristina Palomares Bonache, Jordi Marco Gomez and Xavier Franch Gutiérrez for the opportunity to cooperate in this project and for the dedicated guidance and counselling during the complete development process of this master thesis.

List of figures

List of tables

1. Introduction

This section is an introductory description of the topic, the scope and the work methodology plan for this master thesis, which is titled "*Automated Similarity Detection: Identifying Duplicated Requirements"*.

1.1. Project description and motivation

Artificial Intelligence (AI) is a wide-known computer science area that has experienced exponential growth both in the research field and in real use-case applicability. This computational representation of human cognitive knowledge can be used in many different areas of application, according to the features and the main goals of these fields. Two of the most known areas of AI are Machine Learning (ML) and Natural Language Processing (NLP).

On the one hand, ML is a "general-purpose technology" [1] that uses data and information to learn real-world related knowledge and to improve the reliability of a specific action. Its main application is to use this acquired knowledge to extract autonomous predictions about partial observations of this data. In essence, these systems or algorithms differ from traditional programming schemes due to the results' accuracy improvement based on their own experience.

On the other hand, NLP has a large potential in different applications involving automated, computational processes of all kinds of documents and textual items. This technology applies to the task of developing partial representations of features and rules of natural language based on its textual information, which includes both syntactic and semantic knowledge [2]. The main purpose of this technology is to use this representative knowledge in order to apply automated analysis and generation of text units, such as comprehensive sentences or full documents.

One of the areas of application of AI - and more specifically, NLP and ML - is the Requirements Engineering (RE) field. RE is the set of activities and processes of Software Engineering (SE) focused on the development, analysis, communication, and management of a set of requirements that describes the features of a system [3]. Software development experience in recent years proves that managing and maintaining large sets of requirements have become critical issues. This problem is even more challenging due to the management of a large amount of data and the dimensions that these projects are dealing with nowadays. Whether the analysis and the evaluation of requirements is a tedious, time-consuming task, it is critical that they are carried out with both accuracy and efficiency in any software development project.

Between the main problems of RE, the detection and management of duplicated requirements [4] is highlighted. If ignored, these duplicated items may lead to redundancy in the textual information of a project and therefore this may lead to the duplicity of tasks, which are critical issues from the project management perspective. Moreover, the automation of this process and the standardized usage of specific,

accurate tools are still at a state-of-the-art stage. It is difficult to find open source tools and frameworks providing generic, adaptive solutions for duplicate detection and most of them are addressed to a very specific casuistic or use case (UC) [5]. In addition to that, similarity detection algorithms are highly tightened to the quality of the data used for the detection process [6].

This is the starting point of this master thesis: **an analysis of the state-of-the-art of automated requirements similarity detection**, using artificial intelligence techniques, for the detection of duplicates between project requirements. Based on a research of the state-of-the-art, this master thesis is not only a practical evaluation of real duplicate detection algorithms and scenarios in software engineering project requirements. It is also a **software development proposal which integrates different similarity detection techniques** and tunes them based on the defined use case, which is the subject of study of this master thesis.

This thesis will be developed within the OpenReq project [7], an EU Horizon 2020 project whose main goal is *"to build an intelligent recommendation and decision system for community-driven requirements engineering"*. This collaboration allows the usage of real requirements data to evaluate the similarity detection algorithms developed in this project.

The details about the scope and the goals of the project are depicted in the following sections.

1.2. General and specific objectives

Based on the motivations and the field of study introduced so far, the global objectives of the project are listed below.

[O1.] **To research the state-of-the-art of the requirements similarity detection field.** This document must include a summary of the collected information related to the application of similarity detection in the RE field. The achievement of this goal guarantees the necessary input knowledge to propose and develop the algorithmic tool to evaluate a real duplicated requirements detection scenario.

[O2.] **To develop a requirement similarity-detection multi-algorithm tool.** The main core of the developed software must be the technical implementation of the algorithms selected to be evaluated. This development must satisfy usability and evaluation requirements so that it is possible to analyze and to extract conclusions from the achieved work.

[O3.] **To define and to evaluate a real application use case for duplicated requirements detection.** As part of the Horizon 2020 European project OpenReq, this thesis will be tested against a real use case dataset from a company - a set of requirements including duplicated and not duplicated items. Therefore, one of the main goals must be to set up the experiment and data preprocessing, so the data can be tested with the developed tool. The execution

of the experiments must provide tools and techniques to extract empirical results to evaluate at the end of the project, allowing to conclude the thesis with considerations and valid knowledge on the performance and accuracy of the algorithms.

From these general objectives, it is necessary to apply a refinement and a prioritization that leads to the future definition of specific tasks. For this purpose, the list of specific objectives is listed below.

[O1.1.] **To study the current status of similarity detection** in the RE field from a general point of view.

[O1.2.] **To review and to enumerate similarity detection techniques/algorithms**, and specifically the ML and NLP techniques that represent the state-of-the-art of the field.

[O1.3.] **To identify potentially suitable algorithm candidates** for the master thesis and the use case to be validated with.

[O2.1.] **To elaborate a development proposal** for the implementation of the selected algorithms.

[O2.2.] **To integrate the algorithms with a unique tool** to use and to test the different similarity detection scenarios.

[O3.1.] **To evaluate the requirements input data of the algorithms**, in order to guarantee a comprehensive analysis of the results.

[O3.2.] **To optimize and adapt the algorithms** based on the requirements of the use case.

[O3.3.] **To analyze and to prepare a use case dataset** for all scenarios (i.e., all the different similarity detection algorithms).

[O3.4.] **To carry out the experiments** using the developed algorithms.

[O3.5.] **To perform a comparative analysis** between algorithms.

[O3.6.] **To extract conclusions in terms of the reliability** of the results **and the performance** of the algorithms.

1.2.1. Objectives prioritization

The above specific objectives describe the required achievements to be completed during the development of this thesis in order to guarantee the satisfaction of the three main goals. Therefore it is necessary to ensure that all of them are achieved by the end of this master thesis. For this purpose, a prioritization is defined based on a refinement of those objectives that can be adapted and prioritized to be partially achieved. For each one of these objectives, a high priority achievement status and a medium priority achievement status are proposed.

[O2.1.] *"To elaborate a development proposal for the implementation of the selected algorithms."*

a. High priority: to develop a minimum number of two algorithms, which must represent different approaches of the state-of-the-art similarity detection proposals.

b. Medium priority: given the situation when the required amount of time to analyze and develop the first two algorithms is more than enough, it can be considered to add a third algorithm to the developed system.

[O3.6.] *"To extract conclusions in terms of the reliability of the results and the performance of the algorithms."*

a. High priority: to extract reliability results that allow the analysis of the accuracy of the algorithm, without considering the performance or the execution time of the algorithm.

b. Medium priority: to extract both reliability and performance results, performing a complete analysis that evaluates a balance between the efficacy of the algorithms and its efficiency.

1.3. Approach and methodology

The general and specific objectives of this master thesis are oriented to base the development in three main edges. The first one is a state-of-the-art review, which is documented and presented in this thesis document. The second one is the development of a RE system which handles the management of requirements data and the similarity evaluation for the identification of duplicated requirements. The third one is an experimentation stage for a qualitative, comparative analysis between the algorithms.

This document is a report of all the work developed as part of the thesis. Additionally, the software system developed is published on a public Github repository, which is available here: https://github.com/quim-motger/tfm.

The commit history log of the repository can be additionally used to follow the development and the tasks that have been completed. Moreover, the repository includes documentation for developers such as a README file with technical information.

1.3.1. Development methodology

This thesis has been developed following a **Kanban-based methodology** with some general influences from Scrum, which has been proven to be a good approach for projects of this kind of nature (in terms of size and effort) [8]. This decision is justified by three main reasons.

First of all, it seems suitable to propose an agile software development methodology to achieve the goals depicted in this document. Although this project starts from a clear,

specific stage, and the objectives are detailed enough, the research of the state-of-the-art phase will deeply condition the specific tasks that will be done during the technical implementation and the evaluation process (i.e., the number and nature of algorithms to be integrated into the tool and that will be afterwards evaluated). It is necessary to handle certain flexibility in terms of requirements and tasks during the project's development.

Second of all, this methodology aims to provide results in short-term cycles by scheduling fine-grained tasks that guarantee the success of the project's objectives. Following one of the main guidelines of Kanban, which is the visibility and traceability of the tasks, it is intended to provide a dynamic framework that will allow to complete tasks in a short period by identifying, detailing and scheduling them according to the general schedule planning of the master thesis (see section 1.4.1).

Finally, and following with this last criterion of cyclic and iterative results, one of the main goals of this methodology is to hold weekly retrospective and plan meetings, comparable to analogue meetings from the Scrum methodology, with the thesis director. These meetings will allow not only a constant review of the work that has been done but an iterative review of the remaining tasks and the different potential lines of work that might arise during the development of the project.

For achieving and applying the previous methodology plan, it is required to identify and describe the two main artefacts or activities that will guide the project's development.

- **Tasks backlog maintenance.** Based on the Scrum task backlog artefact and the task workflow suggestions from Kanban, a project tasks backlog will be used to identify, classify and organize the development of each of the tasks raised during the development of the project. The specific workflow implementation for this project will be as follows:

  To-Do → Analyzing → Doing → Review → Done

- **Retrospective and plan meeting.** A regular weekly meeting with the thesis director is proposed to keep track of the general progress of the project. This meeting must include some of the general activities of both the retrospective meeting and the sprint plan meeting from SCRUM [9]. Therefore, the length of each sprint is set to 1 week - which is a short period but will be useful to guarantee the achievement of goals and deadlines as scheduled.

1.4. Work plan

To achieve the specific objectives of this thesis, this section introduces the details of the list of tasks to carry out and a proposal of the schedule for its achievement according to the deadlines of the project. For this purpose, risk identification and mitigation proposals are also introduced.

1.4.1. Stage description and tasks

The work plan is based on a 4-stage plan where each task is defined and assigned with a specific effort estimation. This effort takes into consideration the total number of hours of the project (~300h) and the number of weeks from the beginning of the project (September 16th) to the last submission deadline (January 8th). Consequently, the plan is based on 17 weeks between the beginning and the end of the project. It is necessary to try to keep a balance between the amount of effort required in each week, which must be a value between 15*h* and 20*h*.

1. **Plan**. The main goal of these tasks is related to defining a project proposal and to define the main features of its development before starting to develop the core part of the master thesis.

   Tasks

   [P1.] To perform an initial, generic research about the area of interest of the master thesis *(25h)*.
   [P2.] To define the topic and scope of the project *(5h)*.
   [P3.] To identify generic and specific goals *(5h)*.
   [P4.] To describe and to justify a work methodology *(5h)*.
   [P5.] To elaborate a schedule planning, according to deadlines and available resources *(12h)*.
   [P6.] To identify risks and to propose mitigation plans *(3h)*.
   [P7.] To structure the thesis document and the periodical deliverables *(5h)*.

   Total number of hours = 60 hours

2. **Research.** Includes all the tasks related to the state-of-the-art review, as well as the study and the analysis of ML/NLP techniques, algorithms and technologies for the similarity detection field.

   Tasks

   [R1.] To define a methodology to conduct the research *(15h)*.
   [R2.] To research the state-of-the-art literature of the automated similarity detection field *(20h)*.
   [R3.] To study and analyze a selected subset of algorithms and techniques of interest *(15h)*.
   [R4.] To identify a set of potential similarity detection algorithms to be developed *(8h)*.
   [R5.] To analyze and to compare the set of pre-selected algorithms *(10h)*.
   [R6.] To select those similarity detection algorithms or techniques for duplicated requirements to be implemented and evaluated *(8h)*.

   Total number of hours = 76 hours

3. **Development.** Having the knowledge and the information extracted as a result of the *Research* phase, it is possible to start all the tasks related to the technical

development and implementation of the algorithms and the final software project.

Tasks

[D1.] To identify and to study the specific NLP/ML technologies and frameworks used in the different algorithms *(15h)*.

[D2.] To implement a Proof-of-Concept (PoC) software for each algorithm and its techniques *(30h)*.

[D3.] To validate and to test the scalability of the algorithms for the general use case scenario *(8h)*.

[D4.] To implement stable, usable versions of the algorithms *(20h)*.

[D5.] To integrate the implementation of each algorithm within a tool for usage and evaluation purposes *(8h)*.

Total number of hours = 81 hours

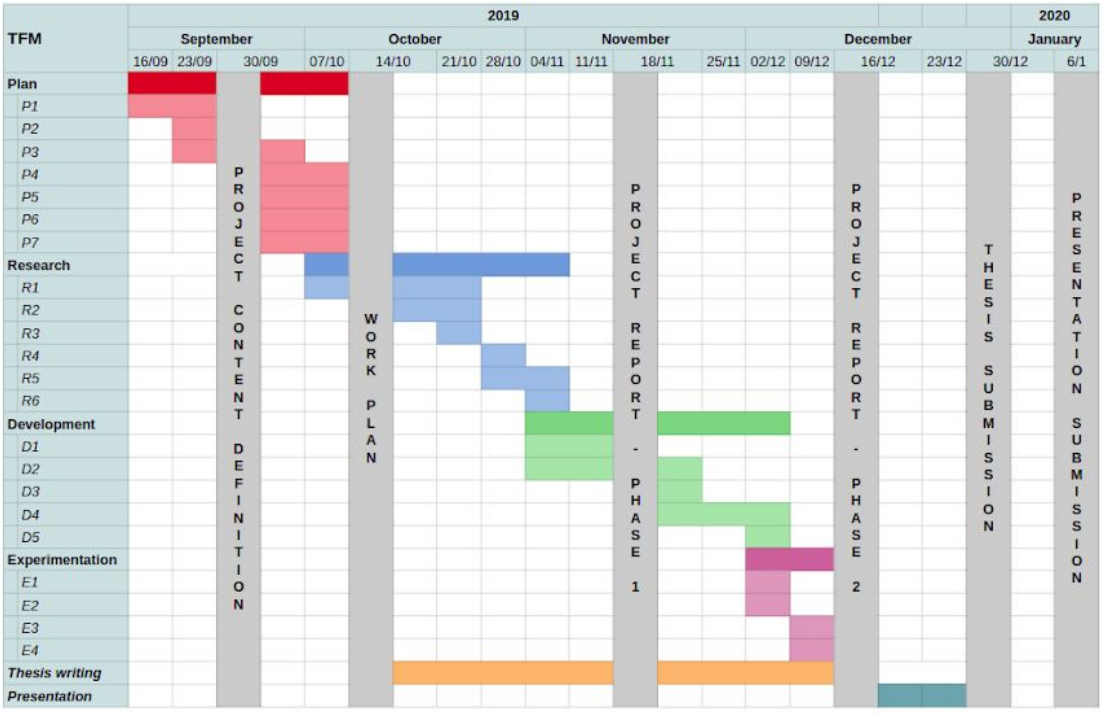

Figure 1. Gantt diagram for the thesis initial work plan

4. **Experimentation.** Finally, the tool resulted from the *Development* stage must be used to run experiments based on a real use case scenario and to extract useful information about the performance of each algorithm.

Tasks

[E1.] To define a set of experiments (i.e., a set of requirements) to test the algorithms *(8h)*.

[E2.] To describe the measures and statistics to use for comparison analysis, which should be useful for both an accuracy and a performance analysis *(4h)*.

[E3.] To run experiments and to collect results *(10h)*.

[E4.] To analyze collected data and to extract final conclusions *(10h)*.

Total number of hours = 32 hours

Additionally, there are two tasks related to non-technical tasks of the project: the *Thesis writing (40h)* and the *Presentation (10h)* elaboration. Together with the 4-stage work plan depicted above, the total effort planned to dedicate to the project is **299 hours**.

The schedule plan of the stages and tasks detailed in the previous paragraphs is presented in Figure 1, which is a Gantt diagram representation of the schedule plan per week, from the beginning of the project until the deadline of the project. For simplification purposes, its granularity is defined at week level. However, the work plan, the distribution of tasks and the number of hours required for each task consider weeks with 5 working days and the distribution of workload considering holidays, like Christmas time period.

Notice that the submission deadline for the master thesis is on December 31st. The writing of the final document is planned to be done simultaneously and progressively during the development of the project.

1.4.2. Risk management

In this section, potential risks that might arise during the development of the project are identified, as well as a proposal of mitigation activities for reducing the impact of these risks in the achievement of the goals described in section 1.2.

[R1.] **Deviation of the original schedule.** It is possible that the estimated time resources for each task may not always be accurate enough, leading to a delay in the completion of tasks and as a consequence a delay on finishing stages on time to keep the project on track.

- *Mitigation*. As a preventive measure, to elaborate a stage plan assigning timing resources in a preventive way, dedicating a ratio of extra hours per task and a time period at the end of the project to correct possible delays with respect to the original plan. If this is not enough, a prioritization of specific tasks that will not compromise the achievement of the main goals of the project will be proposed.

[R2.] **Unexpected obstacles in developing a similarity detection algorithm.** The most critical part of the project is all problems related to the implementation and development of the similarity detection algorithms identified and selected during the *Research* phase. Some of these algorithms may be complex to develop due to different causes, like a lack of knowledge of a specific technology, or the raise of issues or requirements not identified during the state-of-the-art analysis.

- *Mitigation*. To analyze in iterative cycles (i.e., during retrospective meetings) the algorithms development viability and evaluate if necessary the possibility to discard or to change the requirements of the project.

[R3.] **Unexpected results or difficulties in the use case evaluation.** It is possible that an algorithm's execution leads to unexpected problems for evaluation purposes, such as a bad estimation of the required resources for its execution. These resources can be conceived as technical resources (i.e., enough RAM) or human/time resources (i.e., its development is more complex than expected, or a single execution requires too much time to be evaluated).

- *Mitigation*. As a preventive measure, to dedicate specific effort during the state-of-the-art analysis to understand the nature of the algorithms and to identify the requirements for its execution. If this is not enough and the risk is materialized, an alternate approach would be to reduce the size of experimentation and try to apply a scalar analysis for the real use case application.

1.4.3. Time plan deviations and mitigation techniques

During the final stage of the development of this master thesis, it has been necessary to extend 1 week the *Experimentation* stage to finish task E4. This has led the project to an extension of the *Thesis writing* stage until the deadline of the thesis submission (December 31st) and an extension of the *Presentation* stage until the deadline of its submission (January 8th).

During the *Plan* stage, it was proposed to design a preventive schedule as a mitigation technique for the possible risk materialization of a delay in the achievement of some of the tasks. This preventive schedule was designed to finish the whole master thesis project by December 16th - 2 weeks before the real deadline of the project. It was decided to apply this preventive mitigation technique to avoid that a small schedule deviation could lead to unaccomplished tasks or goals, which would have been a major impact on the thesis development. Figure 2 shows the performed schedule according to this minor deviation.

Therefore, thanks to this preventive mitigation risk technique, and based on the analysis of these minor time deviations, it can be concluded that the satisfaction of goals has not been compromised and the thesis has been successfully finished by the time of submission.

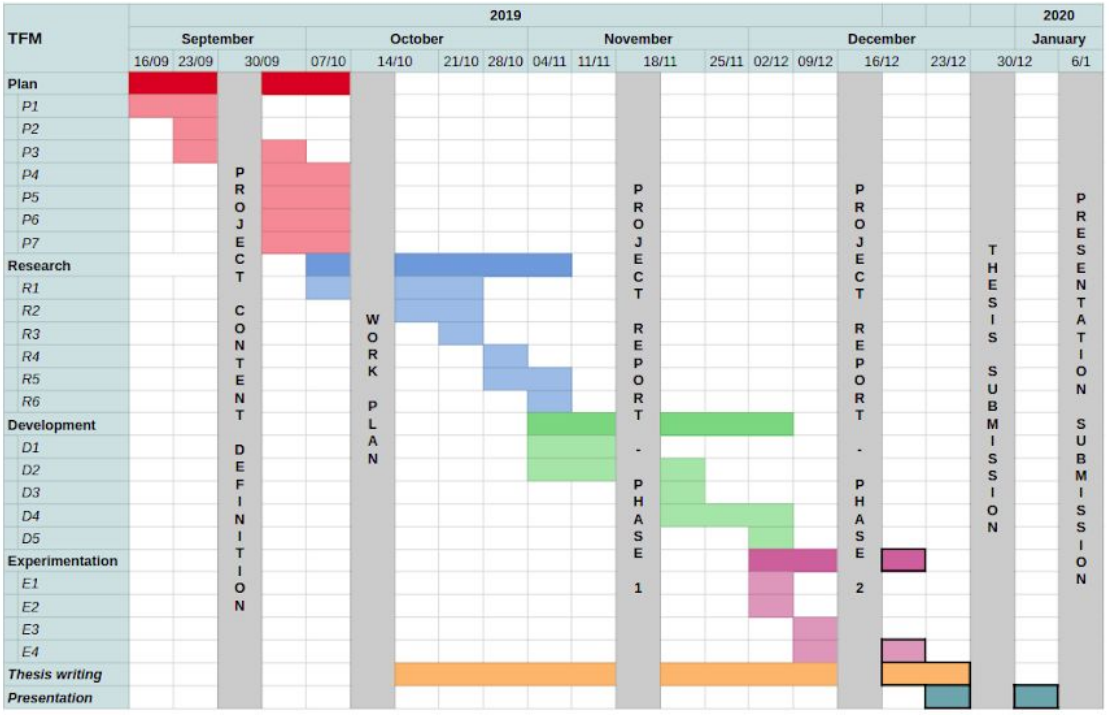

Figure 2. Final Gantt diagram after applying the risk mitigation techniques

1.5. Summary of obtained products and results

In this section, a brief summary of the obtained products and results during the development of this master thesis are provided. Notice that this section does not refer to the organization of this results in this document. This is addressed in section 1.6.

- **Project task backlog.** A Trello board used for task administration and task refinement, following the Kanban guidelines described in the work plan report.
    - *Reference*: https://trello.com/b/4q7jVgd5/master-thesis
- **Literature review results.** After a data synthesis and extraction process, a comparative analysis with general state-of-the-art considerations is presented.
- **Requirements Similarity system.** Published in a Github repository that includes all source code of the Requirements Similarity system and requirements data artefacts used for the system development.
    - *Reference:* https://github.com/quim-motger/tfm
- **Experimentation results.** Experimentation data including reliability and performance results, both qualitative and quantitative, to evaluate and to compare the implemented similarity detection algorithms.

1.6. Thesis organization

Based on the organization of this document, a general overview of its content and structure is provided.

Chapter 2. - Literature research process. Depiction of a systematic literature review process, including a protocol description and the report of the research process itself.

Chapter 3. - Algorithm evaluation & selection. The report of the results obtained during the data extraction & synthesis steps, using the information of the literature review. This leads us to the algorithm comparative analysis and the selection process for future development.

Chapter 4. - Requirements Similarity system overview. A depiction of the requirements and the features of the Requirements Similarity system design and development.

Chapter 5. - BM25F algorithm development. A description of the BM25F approach for duplicated requirements detection and the development process.

Chapter 6. - FE-SVM algorithm development. A description of the FE-SVM approach for duplicated requirements detection and the development process.

Chapter 7. - Experimentation & comparative analysis. It includes the experimentation set-up and the analysis results of the execution of each algorithm.

Chapter 8. - Conclusions. Final conclusions of the thesis development, including a proposal of future lines of work.

More details about each section are provided in each chapter of this thesis.

2. State-of-the-art review

In the following sections, the details about the research method used to describe the state-of-the-art of the similarity detection techniques in duplicated text detection are introduced.

2.1. Definition of the research method

First of all, it is necessary to define a protocol to carry on the research. In order to avoid a vague research methodology that could lead to poor results, it is aimed to define a review method based on a systematic review. Using this guidance, it is ensured a thorough literature review of the field of interest (i.e., similarity detection in natural language texts) of significant scientific value, which is used in this thesis as the foundations for the main developed work.

For this purpose, it is proposed to follow the guidelines of **B. Kitchenham's systematic review methodology** [10], which is focused on applying this review process in the software engineering research field. These guidelines include a series of well-defined stages and processes for planning, conducting and reporting the review.

Therefore, it is proposed to design and implement the following steps for the systematic review, based on Kitchenham's proposal.

1. **Planning the review** (see section 2.2)
    a. Identifying the need for a review
    b. Specifying the review questions
    c. Developing a review protocol
2. **Conducting the review** (see section 2.3)
    a. Identification of the search
    b. Selection of primary studies
    c. Quality assessment study
    d. Data extraction and data synthesis
3. **Reporting the review** (see section 3)

2.2. Planning the review

The following subsections describe the three main steps of the systematic review planning. In this stage, the scope of the research is refined based on the scope of this project by establishing the general requirements of this task (*i.e.*, what was to be researched about or where it is going to be looked for the required knowledge) and all related details about how to perform this review.

2.2.1. Identifying the need for a review

As a first step, it is necessary to justify the need for a systematic review in the similarity detection field for this master thesis. This can be related to the objective *O1*:

[O1.] To research the state-of-the-art of the textual similarity detection field.

Although this state-of-the-art research could be executed following alternative methods to the systematic review, this approach is justified for two reasons.

The first one is related to the requirements and specific objectives of this project. The purpose of developing and evaluating specific similarity detection algorithm implementations is to provide empirical demonstrations of the most important proposals in identifying paraphrase textual units. Furthermore, and as a contribution to the field, it is important to evaluate how these approaches behave in the requirements similarity detection field. It is required to ensure that this review is thorough enough to provide a general overview of the main proposals considered as suitable for solving this problem. This output can then be used as an input to choose specific implementations to develop and to evaluate in the scope of this master thesis.

The second one is related to the available literature in textual duplicate detection. Kitchenham's methodology states the importance of looking for any systematic reviews available on the field, which in case of existing would undermine the need for performing a systematic review for this master thesis. Therefore, the first step is to focus on looking for already available state-of-the-art reviews.

This research is focused on looking for any review related to the similarity detection field using Natural Language Processing, Machine Learning or general Artificial Intelligence techniques. To increase the results of the research, it is not only focused on systematic reviews but in any kind of research regardless of the methodology.

The following databases are used to look for literature review of the field of study:

- Scopus [11]
- ACM Digital Library [12]
- IEEE Xplorer [13]
- Science Direct [14]

Based on the target of the literature review, a search string composed of three blocks of data or information is proposed. These blocks include the following topics:

1. **Similarity detection field.** Any match with "*similar**", "*duplicat**" or "*paraphras**" is used. These are three of the main synonyms used to refer to a pair of texts which may be considered as equivalent.

2. **Artificial Intelligence technologies.** The search must be restricted to the detection of similar textual items using AI technologies, with a special emphasis in NLP and ML techniques, which represent the main approaches for this issue.

3. **State-of-the-art review.** In this stage of the systematic review, it is necessary to focus only on papers and publications which contribute to providing a detailed state-of-the-art analysis. Therefore, it is important to use terms such as "*review*" or "*state-of-the-art*" and "*state of the art*" as part of the search. This will guarantee that one of the main goals of the results in the search is focused on providing this state-of-the-art as an output of the publication.

Consequently, the following search string is proposed:

```
(similar* OR duplicat* OR paraphras* ) AND ( "natural language"
 OR "machine learning" OR "artificial intelligence" OR "AI" OR
    "NLP" OR "ML" ) AND (review OR "state of the art" OR
                        “state-of-the-art”)
```

This search is applied to the title and the keywords of the publications - this will ensure a minimum noise on the results which are not specifically focused on these three main blocks.

These are the results:

- Scopus - 1 result.
- ACM Digital Library - 1 result.
- IEEE Xplorer - 0 results.
- Science Direct - 0 results.

The publication reported by Scopus [15] is related to automated text generation focused on the summarization of user reviews. On the other hand, the publication reported by ACM Digital Library [16] is focused on a state-of-the-art analysis for sentiment scoring via a paraphrase text generation algorithm. Both publications are out of the scope of this thesis.

After a general overview of these results, it is possible to conclude that none of the two results is related to the paraphrase detection field in natural language texts and hence they can be excluded.

As a conclusion, there are no publications providing a detailed, structured analysis of the state-of-the-art techniques for similarity detection between natural language text pairs. Therefore, it is confirmed the need for a systematic review as the first step of this master thesis.

2.2.2. Specifying the review questions

Kitchenham's methodology states the need for defining three elements of the research to help to design and to define the review scope. Applied to the software engineering field, these items are:

- **Population.** It refers to groups or agents (subjects) that are affected by the intervention of the review question. This thesis, and due to the fact that natural language paraphrase detection is a wide application area, is generally focused on Software Engineering teams (developers, team managers...), which are the agents interested in using this automated techniques for solving the problem addressed in this master thesis.
- **Intervention.** It applies to the technologies used to address a specific issue. The research is focused on the automated similarity detection techniques and the algorithmic approaches using machine learning, natural language processing and artificial intelligence techniques in general.

- **Outcome.** It relates to factors and output data which are relevant to evaluate the quality of a specific solution and to compare them. For evaluating the quality of the developed algorithms, the research is focused on the accuracy and the performance of these solutions.

Using these three elements as an input, the systematic review is focused on solving two related research questions:

1. How does the software engineering community handle automated similarity detection between natural language text pairs using AI (*i.e.*, NLP and ML)?

2. Which are the results of these general approaches in terms of accuracy and performance and which are considered as the best approaches from a qualitative point of view using these indicators?

2.2.3. Developing a review protocol

Once the basis of the review has been established, the next step is to define a practical review protocol to be applied when conducting the search. This synthesizes which data to look for, the filters to apply to the result data, and how to extract and analyze the information of each publication from a practical point of view.

2.2.3.1. The search strategy: search & selection procedures

As it has been done to justify the need for a systematic literature review, the search strategy focuses on answering the questions *what* (*i.e.,* which data does the research aim to look for) and *where* (*i.e.,* which databases are going to be used).

- The selected databases are the same ones used in section 2.2.1: Scopus, ACM Digital Library, IEEE Xplorer and Science Direct. These are publicly well-known databases which cover a significant amount of conferences, journals and other publications, with a special emphasis in the computer science field.

- The search string is composed of two of the three main blocks of the one used in section 2.2.1, removing the part related to the state-of-the-art review:

```
(similar* OR duplicat* OR paraphras* ) AND ( "natural language"
 OR "machine learning" OR "artificial intelligence" OR "AI" OR
                        "NLP" OR "ML" )
```

Once the required input data for the research has been selected, a review protocol can be refined to look for and to filter the research results. This protocol is proposed to follow three steps:

1. To search in the databases using the search string defined above.

2. To merge the documents into a single repository and to remove duplicates (*i.e.,* publications found in more than one database). For this purpose, and for future management tasks related to the literature documentation, it is proposed to use the Mendeley tool [17], a research documentation reference manager.

3. To filter the documents following a study selection criteria. This procedure is proposed to follow the next stages.

    3.1. First, to filter publications by title, removing all results that are clearly out of the scope of this research.

    3.2. Second, to filter by reading the abstract of the publication.

    3.3. Third, to filter by giving a general overview of the document, also known as *skimming*. This includes reading some of the main important sections (i.e., introduction and conclusions) or evaluating the general structure of the paper.

    3.4. Fourth, to give a full reading to the document

2.2.3.2. Data extraction & synthesis strategies

Topic	Questions
The domain of the proposal	Is it a domain-specific proposal? If it is, which domain does it apply to?
Objects of similarity detection	Which are the subjects of the similarity detection (full documents, short texts, sentences...)? Does the proposed methodology apply to a specific textual entity?
Similarity algorithm description	Does it include a sequential description of the technical process? Is it detailed enough to reproduce?
NLP preprocessing	Does the algorithm include an NLP preprocessing step? Which are the tasks related to natural language preprocessing of the data?
Similarity functions	Are any word-to-word similarity functions used? If so, which ones?
Machine learning classification process	Does the algorithm include a classification process? Which kinds of features are used (semantic, lexical, syntactical…)?
Output results	Is a similarity score available for each pair of compared text items? Is there an implicit/explicit classification result (*i.e.,* are duplicates retrieved by the algorithm itself, or a threshold score must be used)?
Frameworks and external tools	Is there a list/reference to NLP/ML frameworks and third-party tools used for the similarity detection process? Are they free-to-use?
Experimentation	Is the algorithm tested with real-data experiments? Is data available? Which kind of data is used (type, volume…)?
Results	Is there any reference to results in terms of accuracy? Reliability? Execution time? Used hardware resources?
Evaluable tools	Is the proposal distributed as a tool or piece of code which can be tested?

Table 1. Data extraction template for document reviewal

A data extraction template (see Table 1) is proposed to fulfil for each document of the systematic review. The purpose of this template is to provide a general, structured analysis of the most relevant issues of each publication.

2.3. Conducting the review

This section describes the process of the conducted review and all the decisions that have been made along this process, in correlation with the previous systematic review plan. It also collects the problems and difficulties found during the systematic review process.

2.3.1. Conducting the research

This section relates to steps 1 and 2 of the review protocol: "*To search in the databases using the search string defined above"* and "*To merge the documents into a single repository and to remove duplicates"*.

Two main problems were faced during the search:

- Due to the complexity of the search string, which uses a logical combination of union and intersection logical operations between terms, it was required to adapt the search string to each consulted database. Each document search engine uses its own syntax to define searches and these logical operations between strings and hence it was necessary to adapt and test each one to guarantee the results were applying to the defined criteria.

- As introduced in section 2.2.3, it is necessary to identify and remove duplicated results from the search in order to avoid redundancies. Sometimes publications metadata or document variations present some anomalies making it difficult to automatically detect these duplicates. However, using the previously mentioned *Mendeley* tool, it is also possible to detect documents which are not exact duplicates but have a high probability of being replicates. This feature was used to solve this issue.

After facing the first issue, the search was conducted, retrieving the following results:

- Scopus - 193 results
- ACM Digital Library - 121 results
- IEEE Xplorer- 38 results
- Science Direct - 6 results

The previous list only indexes individual results for each database. The second step of the review protocol was to use the reference manager to remove duplicates and nearly duplicates, which relates to the second problem faced during this step.

- Total nº documents = 358
- Duplicates = 48
- Almost duplicates = 11
- **Final nº documents =** 358 - (48 + 11) **= 299**

At the end of step 2, the research has achieved a total number of 299 publications to be reviewed in the following stages.

2.3.2. Selection of primary studies

This section relates to step 3 of the review protocol: "*To filter the found documents following a study selection criteria"*. This phase is composed of 4 *sub-steps* where each one acts as a filter to reduce the number of documents based on non-relevant research studies identification.

The first step is to **filter by the title** of the publication. After this filter, the number of documents is reduced from 299 to **74**, which means a total of 225 papers are removed from the manager tool. The main reasons for discarding these documents are covered in the following list:

- An important number of papers are focused on the medical field - specifically, they address detecting similarities and replication patterns using artificial intelligence techniques, such as parallel patients, or disease diagnosis by medical recognition. All papers focused on different areas than natural language similarity are rejected.
- Some of the publications are focused on grammar and syntax knowledge of non-romance languages like Hindi or Chinese. The rules and techniques of natural language are highly coupled to the main characteristics of the language itself. Therefore, these publications are excluded and the research is focused on those using techniques for the English language.
- Although matching the search string and none of the above restrictions, some titles suggested that some of the works were focused on solving different problems that are out of the scope of this master thesis - for instance, language translation. These works are also removed from the article repository.

All works which title does seem to focus on the research field and do not satisfy any of the restriction criteria commented above are passed to the next step.

The second step is to **filter by the abstract** by reading it carefully and acquiring a deeper understanding of the subject that the information provided by the title, which sometimes might be vague or imprecise. In this second step, the number of works is reduced from 74 to **34**, discarding a total of 40 works due to reasons like the ones listed below:

- Some abstracts give a deeper knowledge on some of the filtering criteria used on the previous step, like out of field subjects (*i.e.,* an abstract introduced that the work was focused on web-service similarity) or the use of different languages (*i.e.,* Turkish).
- Some works that study the text-similarity are not focused on the semantic dimension, but on other criteria such as the authorship of a piece of text. Therefore, they target the study of properties and other criteria which are of no interest for the scope of this thesis

The third step is based on applying a **skimming** or general overview evaluation of the whole work, reading some of the main parts and studying the general structure and main features of its contributions. In this step, the number of documents is reduced from 34 to 15. Listed below are some of the filter criteria used in this stage:

- Non-relevant contributions or out-of-date approaches that are outdated by more recent works are removed. By applying this simplification, redundancy is decreased.
- Works claiming poor results or a lack of achievement of their main goals.
- Some works are focused on studying and analyzing the word-to-word similarity evaluation. Technical details and the study of word-to-word similarity are assumed to be already acknowledged and this master thesis focuses on providing solutions for the similarity detection between text pairs, specifically NL requirement pairs.
- Some of the general approaches are ontology-based, which means they require an explicit, modelled knowledge of the domain of the input data to detect similarities. It is decided to discard this kind of solutions from the scope of this thesis. This is justified due to the fact that the data of the validation use case of this project does not contain any kind of explicit ontology-based domain knowledge, and it was not possible to achieve it.
- Solutions and algorithm proposals with a clear lack of details for analyzing and reproducing the solutions by developing them are also removed.

Finally, a **full reading** filter is applied for each publication. As a final result, **12** of the last 15 works are selected. The last removals are justified below:

- A paper was too focused on plagiarism and it was more an index of tools and frameworks than a proposal or a technical description of a similarity process.
- A paper was too focused on a tool portfolio and although it provided results and an overview of different algorithms and tools, it did not introduce further details that were necessary to the research.
- A paper required a first process in which training data required chunk identification. Moreover, it did not include enough detail for all features used in the feature extraction process (a total nº of 247 features).

Figure 3 provides a general overview of the systematic literature review process, from the search of the identified databases to each one of the filters applied in this step.

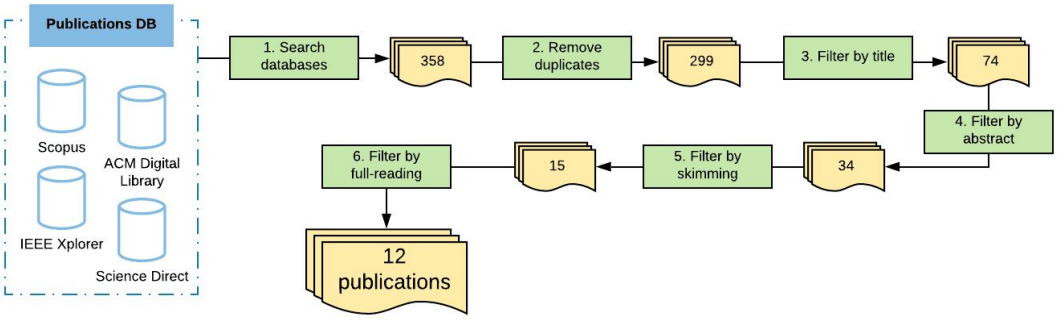

Figure 3. Summary of the Systematic Literature Review process

The following list enumerates the publications extracted from the systematic literature review whose algorithms are going to be evaluated.

- ***P_1***. *"Sentence similarity based on support vector regression using multiple features" [18]*
- ***P_2***. *"An approach to detecting duplicate bug reports using natural language and execution information" [19]*
- ***P_3***. *"Towards more accurate retrieval of duplicate bug reports" [20]*
- ***P_4***. *"Statistical Analysis of ML-Based Paraphrase Detectors with Lexical Similarity Metrics" [21]*
- ***P_5***. *"Machine Learning Based Paraphrase Identification System using Lexical Syntactic Features" [22]*
- ***P_6***. *"FBK-HLT-NLP at SemEval-2016 Task 2: A Multitask, Deep Learning Approach for Interpretable Semantic Textual Similarity" [23]*
- ***P_7***. *"Paraphrase identification on the basis of supervised machine learning techniques" [24]*
- ***P_8***. *"Learning Term-weighting Functions for Similarity Measures" [25]*
- ***P_9***. *"Robust semantic text similarity using LSA, machine learning, and linguistic resources" [26]*
- ***P_{10}***. *"A grammar-based semantic similarity algorithm for natural language sentences" [27]*
- ***P_{11}***. *"Detection of duplicate defect reports using natural language processing" [28]*
- ***P_{12}***. *"Paraphrase Recognition via Dissimilarity Significance Classification" [29]*

2.3.3. Data extraction & synthesis

The previous steps of the systematic literature review lead us to an output of 12 relevant documents to answer the research questions. To complete this stage of the master thesis, it is needed to extract all required data using the data extraction template described in Table 1. These questions and topics will allow us to analyze and structure all the acquired knowledge from the literature review in a systematic, organized way.

The results of this last step are introduced in the following section as a similarity detection state-of-the-art report.

3. Systematic literature review results

The following sections provide a thorough depiction of the state-of-the-art of natural language similarity detection processes using NLP and ML technologies.

First of all, a general overview of the different algorithmic approaches is presented. The topic is introduced by analyzing how the most up-to-date solutions try to solve the evaluation of the similarity between two pairs of texts. This includes understanding what kind of data do they work with, what are the results, and what kind of technologies are used to evaluate this similarity.

Second of all, NLP and ML technologies mentioned in these contributions are introduced, and it is analyzed how do they apply in paraphrasing detection. For the general algorithmic solutions, it is studied which kind of NLP techniques are used and how do they help in this process. Additionally, those solutions that require a classification process for duplicate/not-duplicate discernment are highlighted

Third of all, it is important to understand which kind of metrics are used for evaluating the efficacy and the efficiency of these solutions.

Finally, an analysis of the algorithm is provided and the selection of the final algorithms to be developed is justified.

3.1. General algorithmic approaches

The first step to understanding how the Software Engineering community is handling paraphrasing detection is to try and classify the different algorithmic solutions in general categories. From this general structure, it is possible to depict the details, the context and the technical specifications of each of these approaches.

After a full reading of the documents, it is proposed a **classification in 3 main general approaches** for evaluating the similarity between natural language text units. This classification is similar to the one proposed by Fu et al. in [18], but it is intended to provide a more abstract proposal which can be understood without specifically referring to sentence-similarity, as [18] does. Additionally, further details related to the information from other contributions are provided.

1. **Align-based similarity**. Generally, alignment approaches analyze pairs of documents or sentences individually, as isolated pairs of text units. From this analysis, an alignment approach is carried out: words and extracted features from each text unit are aligned and compared with the other pair, and for each comparison, a score or a quality coverage is assigned. According to these results, it is possible to determine whether a pair of texts is similar or not.

2. **Vector-based similarity.** These approaches transform each document or sentence into a *bag-of-words* vector. This unidimensional data structure allows analyzing a text unit as a set of words or *tokens*, where each word has a frequency and a position in the text unit. The global representation of all text

units, which is usually called the *corpus*, can then be analyzed as vector-space representations. As a consequence, vectors representing each text unit are used for computing a similarity measure for each pair of documents or sentences.

3. **Feature extraction with supervised classification.** These machine learning approaches define a two-step process. First, for each pair of text units, a series of features are extracted using different NLP techniques. These features usually include both lexical and syntactic analysis (see section 3.2.2 for more detail). Second, these featured pairs of text units are used to apply a supervised classification process. In this process, a set of labelled data (*i.e.,* pairs of documents/sentences labelled as *duplicate* or *not-duplicate*) is used to train a classifier, which after training can be used to predict whether a new pair of text units are duplicates or not.

The solutions described above share a significant amount of processes and techniques, especially those related to NLP, similarity measures and feature extraction processes. These techniques are analyzed in section 3.2.1.

From this general classification, the first feature to use as a criterion to classify and select the algorithms to be developed is defined as follows:

- **F_1: algorithm type.** Algorithm classification in 3 general categories: align-based algorithms (AB), vector-based algorithms (VB) and feature extraction with supervised classification algorithms (FE). This criterion is not used to exclude but to classify and encourage a comparative analysis between different algorithmic approaches

Another noticeable detail from a general point of view is that similarity detection between text units is highly coupled with **the context of the similarity evaluation** problem it is required to solve. This context can be expressed in terms of which kind of data is evaluated and what kind of similarity is explored.

With respect to the data, the main contributions in the field mostly focus on similarity detection between isolated, individual pairs of sentences. This implies that the similarity evaluation result will not depend on the global dataset or *corpus* used for the analysis. In some situations, this might be an advantage, as the quality of the algorithm does not depend on the quality or the amount of available data - which is also one of the requirements for supervised classification processes. However, ignoring the context and the implicit knowledge of the domain of the documents can also reduce the quality of results, especially when working on a very specific domain.

Context also determines how the similarity detection problem is applied to a specific use case. Some approaches like [19] or [20] are focused on retrieving duplicate issues (*i.e.,* requirements) in a bug repository. These repositories usually contain large amounts of data, and the duplication retrieval problem applies to evaluate whether a new issue may be a duplicate to another one already reported. In these situations, it is additionally required to provide a specific evaluation algorithm for the described use

case. Therefore, quality results and execution time of the proposed solutions must be suitable for the use case.

Two features are defined from these observations:

- **F_2: sentence / full-text.** Whether the algorithm is focused on comparisons between pairs of sentences (S) or full texts or documents (D).
- **F_3: requirement focused.** The contribution is focused on a specific text entity based on a real use case from the RE field.

With respect to the **kind of similarity**, this relates to understanding what *similarity* means. Some approaches focus exclusively in similarity as the level of paraphrase between two sentences or texts. This means at which level two text units are expressing the same idea. Other approaches focus on analyzing if two text units are talking about the same topic, with almost no differences. This can be summarized as the difference between lexical and syntactic similarity. An example can easily illustrate the difference between these two approaches. Let's consider a pair of sentences:

S_1 *: My cat is following your dog.*

S_2 *: Your dog is following my cat.*

If focusing on the semantic meaning of each sentence, it can be easily concluded that S_1 and S_2 are describing exactly opposite situations. A syntactic analysis would easily reflect that in S_1 *'my cat'* is the subject and *'your dog'* is the object of the *'following'* verb, while in S_2 is exactly the opposite. Therefore, it could be concluded that S_1 and S_2 are not duplicated. This would be considered a syntactic similarity analysis.

However, if focusing on a lexical analysis, it would be recognized that S_1 includes two nouns/entities (*'my cat', 'your dog'*) and one verb/action (*'following'*), which are a perfect match with S_2. As a conclusion, from this point of view, it could be considered that S_1 and S_2 are duplicates. This would be considered a lexical similarity analysis.

Some approaches like [21] focus exclusively on lexical similarity. Other contributions like [22] consider both lexical and syntactic analysis in their evaluations. It is relevant to differentiate these two similarity approaches and how they affect the quality of the results in the different proposals.

Based on these differences, two additional features are extracted:

- **F_4: lexical similarity.** The algorithm uses lexical analysis techniques as part of the similarity evaluation.
- **F_5: syntactic similarity.** The algorithm uses syntactic analysis techniques as part of the similarity evaluation.

To conclude with this general analysis, it is necessary to state that in order to develop and analyze a specific algorithm, it is required that the publication it is based on provides enough details with respect to its process and the specific technologies used so it can be reproduced and adapted. For this purpose, an additional feature is added:

- **F_6: reproducible algorithm.** The algorithm is detailed enough to replicate and develop it step by step.

Table 2 summarizes how the criterion mentioned above match with each of the publications extracted from the literature review.

feature / P_i	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
F_1	AB+ VB	VB	VB	FE	AB+ FE	FE	FE	AB	FE	AB	VB	VB
F_2	S	D	D	S	S	S	S	S	S	S	D	S
F_3		✓	✓								✓	
F_4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F_5	✓				✓	✓				✓		
F_6		✓	✓	✓	✓				✓	✓	✓	✓

Table 2. General features analysis

After a general overview of the similarity detection problem, further details about the technologies used in these algorithm proposals are provided.

3.2. Technologies in text-similarity evaluation

The three main approaches described in the previous sections are general descriptions of a set of processes that basically include processing sentences and transforming them into data structures that can be computationally analyzed. Additionally, these data structures can be used in a supervised classification process in some approaches. This section aims to provide technical details about the processes used in these solutions.

3.2.1. Natural Language Processing in requirements data

Generally, all algorithms studied for this master thesis include some basic preprocessing of the textual data of each text unit, which is later used in the similarity evaluation formula. These basic techniques include some of the most common NLP processes mainly focused on **lexical analysis**:

- **Tokenization.** The process of breaking a sentence or text into an ordered list of words, elements or *tokens*.

- **Stop word removal.** The process of iterating over a list of words or tokens and remove those that are meaningless for future data processing. These *stop words* include articles, prepositions and pronouns, among others, and are usually removed as they do not provide significant meaning to the NLP techniques.

- **Part-Of-Speech (POS) Tagging.** Each token is marked with a POS tag, which is the category of words to which the token belongs to. There are different POS

tags which might be more or less exhaustive, but essentially they classify tokens into nouns, verbs, adjectives, adverbs, etc.

- **Stemming.** It corresponds to the process of identifying the root of the word or token. This process removes suffixes and prefixes of each word, keeping only the root or *stem* of each word. It allows reducing variability between words deriving from the same root.

- **Lemmatization.** It identifies the lemma, *i.e.,* the lexeme, of a specific keyword. Lemmatization looks for the original word from where the analyzed derivative comes from. As it is more expensive from a computational point of view, some approaches like [20] or [21] limitate to using stemming in their data preprocessing, while others like [22] or [23] are based on using lemmas for the similarity analysis.

After the lexical analysis, some algorithms like [22] or [23] apply an additional syntactic analysis, which applies to identify the roles and relations of the different elements that compose each sentence (subjects, objects, direct-object, indirect-object...).

- **Sentence boundary disambiguation (SBD).** Although it is only applied in those algorithms where the input data are not sentences but full texts, SBD is the technique of identifying the boundaries of a sentence inside a text (*i.e.,* its starting and its ending position). This is required when a syntactic analysis is needed, as the algorithm must process sentences individually to apply a dependency analysis.

- **Dependency parsing.** The process of generating a dependency tree of a sentence which is the representation of its grammatical structure. Each node of the tree is a word or a member of the sentence, and the edges are the grammatical relationships between these members.

The output of these processing techniques allows to calculate similarity measures or **alignment** features, based on the intersection of tokens/stems/lemmas, and the syntactic elements of sentences, among others.

In [22] they provide an exhaustive proposal of both lexical and syntactic alignment features that are used in other publications like [23]. Some of these features include matching **n-grams** between text units. N-grams are subsequences of size *N* of a set of words, i.e., of a sentence or text. Matching proposals include computing the intersection between the uni-grams or bi-grams between two text units. This approach is essentially focused on the lexical similarity between two sentences.

On the other hand, syntactic alignment features can also be used in some approaches. These alignments require a sentence-per-sentence comparison. Some examples include a subject-subject comparison or object-object comparison. The premise is to compare and match the grammatical structure of a sentence with its pair.

With respect to **vector-space** representations, approaches like [20] or [25] use specific similarity measures using this bag-of-words representation as an input.

One of the most common vector-space representations is the inverse document frequency (IDF). IDF is a numerical statistic from Information Retrieval (IR) which focuses on emphasizing the relevance of a given word inside a set of documents or corpus. Given a set of documents *D*, a specific document *d* that belongs to *D,* a term *t* of *D,* and the set of documents d_t that contain *t*, the IDF of *t* is defined as follows:

$$IDF\ (t)\ =\ log(\frac{N}{N_d})$$

A high value of IDF is reached by a low frequency of the term *t* in the corpus *D*. This can be understood as a lowly used topic or keyword that only appears in very specific text units, meaning that its appearance is probably significant for that specific text. The result of this process is a vector-space representation where each word is not only represented by its appearance on document *d* but also according to the importance of its frequency in the whole corpus.

There are several functions used for calculating the similarity between these vector-based representations. Two of the most important are:

- **Cosine similarity.** From a mathematical perspective, the cosine similarity is a metric that calculates the angle between a pair of vectors projected in a multi-dimensional space. If applied to the natural language field, these dimensions are the words, and the vectors describe the appearance and occurrence of these words in this vector. The cosine similarity then describes the similarity between documents based on this vector-space representation. The main advantage of this measure in comparison with others like the Euclidean distance is that it does not take into account the size of the document, which might not be meaningful for document similarity.

- **Jaccard similarity.** The Jaccard similarity is a set theory formula based on the intersection between two different sets. Given a pair of sentences and their vector-space representations V_1 and V_2, the Jaccard similarity is defined as:

  $$J\ =\ |\ V_1 \cap\ V_2\ |\ /\ |\ V_1\ \cup\ V_2\ |$$

  As a result, a normalized, size-independent value between [0..1] is obtained. This can easily be used to evaluate the level of similarity between the two vector-space representations.

Some contributions elaborate a deeper proposal by defining an **optimization** process for this vector-based representations. In [20] they define a weighted variance of the IDF approach with a set of free parameters, which can be optimized using a genetic algorithm or another optimization process. For this purpose, it is necessary to prepare a dataset with explicit knowledge about the level of real similarity between pairs of texts (i.e., labelled duplicate/not-duplicate pairs of text units).

Two additional features are defined based on the extracted knowledge analyzed above:

- **F_7: similarity measure.** The algorithm uses a specific similarity measure to provide a score.
- **F_8 optimization process.** The algorithm includes an additional optimization process to the similarity measure.

3.2.2. Machine Learning classification techniques

Beyond the natural language preprocess of the previous sections, some proposals are focused on an alignment and feature extraction process between pairs of text units and a classification algorithm. It is important to mention that all studied classification techniques focus on **supervised classification**, meaning that it is necessary a labelled dataset with a set of pairs of text units and its class - in this case, *D* (duplicate) or *ND* (not duplicate).

The first step of these approaches is based on the feature extraction process. Literature shows a discrepancy between the relevance of using only lexical features, syntactic features or both of them. Proposals like [22] claim that syntactic features do not improve the results concerning using only lexical features. In fact, they claim to get even worse results. However, other contributions like [27] demonstrate the need for using syntactic features to improve the classification quality results.

Additionally, there are also different considerations with respect to the kind of classifiers to use for this purpose. There seems to be a general consensus among the *Support-Vector Machine* (SVM) classifiers, to which some contributions focus on (e.g., [24]). Furthermore, some proposals introduce a multi-classifier approach in which classification is done via a voting-based approach (where the most frequent prediction between the different classifiers is the one reported as correct). This set of classifiers includes Naive Bayes, SVMs and neural networks [22], among others.

Therefore, it is necessary to define an additional feature related to these approaches:

- **F_9: supervised classification process.** The algorithm requires labelled data and requires a supervised classification process to detect duplicates.

Table 3 summarizes the feature-matching process based on the information of this section.

feature / P_i	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
F_7	✓	✓	✓					✓		✓	✓	✓
F_8			✓			✓						
F_9				✓	✓	✓	✓		✓			

Table 3. Text-similarity specific evaluation techniques features

3.3. Data results and evaluation

The evaluation of the results of each algorithm can be analyzed from an accuracy perspective and a performance perspective.

If focusing on the accuracy dimension, two ways of providing results data can be differentiated. The first one relates to the alignment and vector-space approaches. For these proposals, the result of each comparison between a pair of documents or texts $\{d_1, d_2\}$ is a **similarity score**, independently on the way this score is calculated. Higher values of this score represent a higher level of similarity between these documents.

In order to calculate the accuracy of these results in identifying duplicates, there are different proposals based on this output score. In [20], they discuss the usage of the **recall-rate@k** measure. Given a set of documents *D*, and a set of known duplicate pairs $\{d_i, d_j\}$, for each *d* in *D* the *top-k* list of the most similar documents to *d* is computed (i.e., those with a higher score). This experiment is repeated with different values of k=1,2,3...N. Then, for each value of *k*, it is calculated how many of the $\{d_i, d_j\}$ appear on these top lists, which can be translated to an accuracy score at a given *k* value. The main advantage of this measure is that it does not require any other decision-making process, like a classification step. However, it is computationally expensive, as it requires an *all-to-all* comparison between all the requirements of the *corpus*.

Other proposals like [21] discuss deciding a **threshold** value used to split the data. This threshold is used to separate the two classes or predictions: those pairs with a value lower than the threshold are considered not duplicates, while those pairs with a value equal or higher are considered duplicates. This, however, faces the challenge of deciding a threshold, which can be done using different quality criteria.

This approach leads to the second approach: providing a **confusion matrix** about the classification between duplicates and not duplicates. This matrix computes:

- **True Positives (TP):** duplicate pairs predicted as duplicate.
- **True Negatives (TN):** not-duplicate pairs predicted as not-duplicate.
- **False Positive (FP):** not-duplicate pairs predicted as duplicate.
- **False Negative (FN):** duplicate pairs predicted as not-duplicate.

From these metrics, the following statistics can be used to calculate the quality of the results of each algorithm:

- **accuracy =** (TP + TN) / (TP + TN + FP + FN)
    - Computes the overall proportion of correct labelled pairs vs. the total number of comparisons.
- **precision =** TP / (TP + FP)
    - Computes how many of the duplicate pairs detected are indeed duplicates.
- **recall =** TP / (TP + FN)
    - Computes how many of the real duplicate pairs have been detected by the algorithm.
- **f-measure =** 2 * *precision* * *recall* / (*precision* + *recall*)
    - Computes the harmonic mean between precision and recall.

The metric to focus on depends on the use case and the context of the problem to solve. For instance, in [20] they focus on maximizing the precision of their algorithm.

This is due to the fact that they are evaluating the existence of any duplicate candidate when a new issue is submitted to a bug repository, with large amounts of data. Therefore, their goal is to reduce the number of false positives (i.e., wrong duplicates). On the other hand, in [22] they focus on recall, as they carry out an evaluation based on a public dataset from Microsoft developed by [30] with a set of duplicate and not-duplicate pairs of sentences.

A feature related to the accuracy metrics is necessary to perform an analysis and compare the obtained results:

- **F_{10}: efficacy results.** Accuracy metrics are provided.

Concerning **performance metrics**, most of the reviewed contributions do not provide any kind of efficiency measure to evaluate the execution time or the required resources to run the algorithms. The main reason for this is that most of the contributions focus on individual *text-to-text* pair similarity, which usually is a very low time-consuming task. This kind of metrics is only provided when the algorithmic approach includes a classification process or when the algorithm is designed based on a use case which works with large amounts of data.

However, even in these reports, only execution time metrics are provided. More specifically, they publish the required execution time for a specific number of comparisons, which can be converted into the number of *text-to-text* comparisons per second.

Nevertheless, the following feature is proposed:

- **F_{11}: efficiency results.** Execution time metrics (i.e., a subtype of performance metrics) are provided.

Table 4 summarizes the feature-matching process of the data results and evaluation features.

feature / P_i	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
F_{10}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F_{11}			✓		✓							✓

Table 4. Data results and evaluation features

Chapter 7 focuses on experimentation and the metrics used for accuracy and performance evaluation for each algorithm.

3.4. Algorithm selection and technical analysis

Based on the data extraction and synthesis template described in chapter 1 and the reported knowledge in the previous sections, a set of features are proposed to evaluate each algorithm. This approach will help to evaluate a systematic, structured strategy for

algorithm selection, as well as to clearly define each contribution and its interest in the field.

First of all, it is necessary to define how each feature is going to be used to select the algorithms to be developed. Generally, each feature is used either to **classify** the algorithms, to **prioritize** them, or to **remove** them. The list below depicts this analysis:

- **F_1: algorithm type.** Algorithm type is used to classify between the most representative approaches for finding similarity between natural language text pairs. It is necessary to avoid comparing very similar approaches that do not represent the state-of-the-art of the field.
- **F_2: sentence / full-text.** Similar to F_1, F_2 is used to classify similarity detection processes. Although solutions based on full documents are more suited to this thesis scenario, it is not necessary to remove or prioritize these solutions regarding those based on sentence similarity. Sentence-based solutions can be easily adapted to document scenarios.
- **F_3: requirement focused.** This feature is used to prioritize those algorithms which validation domain is focused on software engineering environments and, to be specific, in the management of requirements.
- **F_4: lexical similarity.** It is used to classify the different solutions, as it is not necessary to focus on a specific kind of similarity. Nevertheless, as stated with F_1, it is necessary to encourage diversity among the selected algorithms regarding the kind of similarity.
- **F_5: syntactic similarity.** Analogue to F_4, it is used to classify the different solutions and to provide diversity in the comparative analysis.
- **F_6: reproducible algorithm.** This feature is used to remove publications from the candidate list. Although algorithms will be adapted and tuned to the defined scenario, it is necessary to have enough details to develop them. Therefore, it is necessary to exclude those that are not detailed enough to replicate.
- **F_7: similarity measure.** Used to classify those approaches based on providing a specific similarity measure or score that can be used to compare different similarity evaluations.
- **F_8 optimization process.** Used to prioritize those solutions that propose a specific optimization process using available labelled data, which is an indicator of potential improvement based on the use case or scenario (i.e., data) used for validation.
- **F_9: supervised classification process.** As labelled data is available for the validation of this master thesis, this feature is used to classify algorithms, and not to discard them.
- **F_{10}: efficacy results.** It is not required that the publications provide efficacy results, but this feature is used to prioritize publications. If efficacy results are available, it is possible to both validate and compare the developed algorithm with the version reported by the publication.
- **F_{11}: efficiency results.** Used to prioritize different approaches analogously to F_{10} and using the same criteria.

The results of this comparison are collected and summarized in Table 5, which summarizes the feature-matching process depicted in Table 2, 3 and 4.

feature / P_i	P_1	P_2	**P_3**	P_4	**P_5**	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}
F_1	AB+ VB	VB	**VB**	FE	**AB +FE**	FE	FE	AB	FE	AB	VB	VB
F_2	S	D	**D**	S	**S**	S	S	S	S	S	D	S
F_3		✓	✓								✓	
F_4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F_5	✓				✓	✓				✓		
F_6		✓	✓	✓	✓				✓	✓	✓	✓
F_7	✓	✓	✓					✓		✓	✓	✓
F_8			✓			✓						
F_9				✓	✓	✓	✓		✓			
F_{10}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F_{11}			✓		✓							✓

Table 5. Selection criteria & data synthesis evaluation

Based on these selection criteria, and analyzing the results presented in Table 5, it has been decided to select, analyze and develop two similarity algorithms based on the following proposals.

[A1] P_3 - *BM25F approach: an extension of an Information Retrieval algorithm.* This similarity detection proposal is focused on detecting duplicates in public issues repositories, which is a common scenario in RE. Their solution is based on an Information Retrieval (IR) measure called BM25F (based on TF-IDF), which they extend with additional lexical metrics and a custom weight optimization process. They claim to improve BM25F traditional results, both in accuracy and performance. This solution is a very good candidate that represents the vector-based approaches, including an optimization process and a focus-oriented solution to requirements similarity evaluation.

[A2] P_5 - *FE-SVM approach: a feature extraction process with data classification.* This proposal is a representation of both align-based and supervised classification approaches. They combine lexical and syntactic knowledge, which are obtained by an align-based evaluation process of different lexical and syntactic features. Additionally, they propose and evaluate the comparison between these sets of features, providing both efficiency and accuracy results of the classification process.

Further details of these algorithms are provided in chapters 5 and 6.

4. Duplicated requirements detection system

This chapter analyzes and describes the requirements similarity detection system to be developed for this master thesis. The architecture, the model and the technical specifications described below are fundamentally based on two edges.

The first one is intrinsically coupled to the research carried out in this thesis. This includes the selected algorithms, which will be developed in chapters 5 and 6, as well as the goals of providing comparative results to evaluate these approaches as representations of the state-of-the-art of similarity detection.

The second one is linked to the OpenReq project, which has already been introduced in section 1.1. In this thesis, a general structure and development requirements are used to apply the research and development work to a specific real use case. This contribution leads to some specifications which are needed to be satisfied in order to apply the results to the available dataset. These specifications, both from a technical and an architecture perspective, are detailed in this section.

4.1. System design and description

The **requirements similarity detection system** is focused on serving three main features:

1. **Requirements data management.** Especially interesting when working with large datasets of requirements, this system must essentially decouple the management of data (i.e., storing, updating and retrieving requirements information) from the comparative and similarity evaluation itself. It is required to provide an infrastructure capable of storing and managing requirements data according to the similarity process requirements.

2. **Requirement pair similarity evaluation.** Each algorithm provides different approaches on how to find duplicates between a set of requirements. However, and focusing on the first research question (see section 2.2.2), functionalities that allow developers to ask for the similarity between two specific requirements are provided, using each one of the selected algorithms.

3. **Optimization, experimentation and comparative analysis.** In alignment with the second research question, it is needed to provide tools and features to perform experiments and to extract qualitative results for each algorithm.

Features 1 and 2 are focused on providing a system that can be used for the final users, i.e., software engineers. Feature 3 is focused on providing results for this master thesis.

4.1.1. General overview

Figure 4 is a general abstraction overview of the different components intervening in this master thesis and the system to be designed and developed. Specific software architecture and technical details are given in section 4.2.

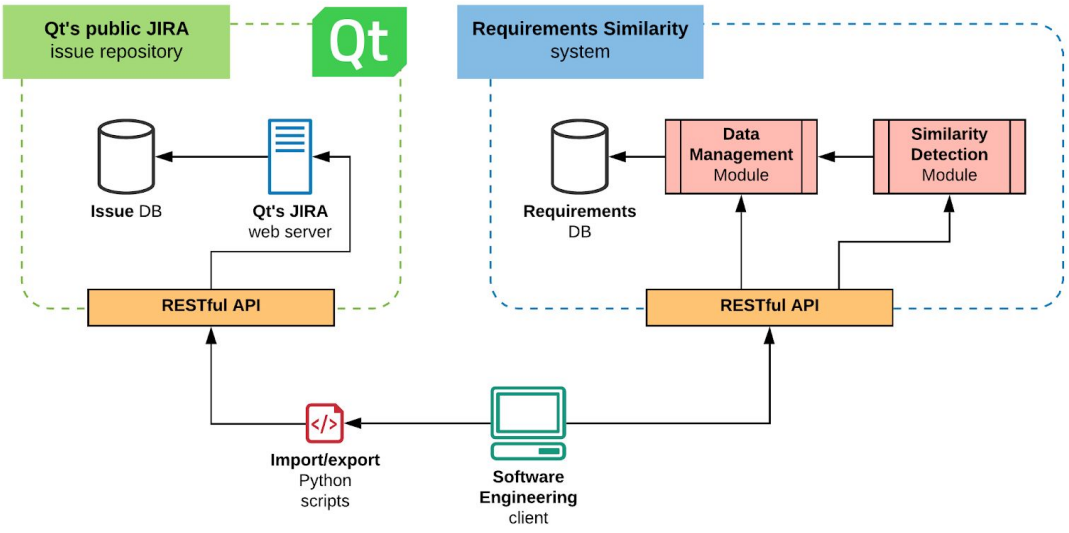

Figure 4. Requirements Similarity system general overview

A general depiction of these components is provided below:

- **Qt's public JIRA issue repository.** The Qt Bug Tracker is a public tool from the Qt Company which allows users to report, look for and discuss bugs, feature improvements or other issues in Qt products [31]. It is built upon Jira[1], an Atlassian development software tool that allows the management and traceability of different kinds of requirements [32]. More details are given in section 7.1.

  From this master thesis perspective, the Qt's issue repository will be handled as a black box in the system from which to get all necessary data to evaluate the algorithms and to carry out the experiments. For this purpose, it is only necessary to evaluate and communicate with the RESTful API, which allows full access to web server data.

- **Import/export Python scripts.** As part of the project requirements, it is necessary to communicate to the Qt's JIRA RESTful API to download requirements data, which will be imported to the system. For this purpose, a set of Python scripts that communicate with the API to export the required information have been developed. More details about this data are given in section 4.1.2.

- **Software Engineering client.** The client in this system is the host machine that uses the Requirements Similarity system. For this thesis, its role is to act as a communication bridge between the data extraction process from Qt's Bug Tracker and the Requirements Similarity system. It is responsible for sending the data and asking for the evaluation of the algorithms.

- **Requirements Similarity system.** This is the core result of the master thesis development stage. A documented, usable tool that allows software engineers

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

to import requirements data and to evaluate different similarity detection algorithms is provided. This evaluation is demonstrated by both experimentation features and evaluation features for specific pairs of requirements.

An abstraction of the system is provided to identify 3 main layers:

- **RESTful API.** The interface of the tool deployed as a web service exposing a RESTful API documented with Swagger UI[2]. This API includes all requests the system exposes and the necessary information to use it by any software client via HTTP requests. This interface provides a unique, uniform layer to access the two main sets of features of the service: the Data Management module features and the Similarity Detection module features.
- **Domain modules.** Submodules of the system handling main business logic, oriented to the different main features of the system.
    - **Data Management module.** Implements all services and domain logic regarding the management of requirements data. This includes all CRUD functionalities (*Create, Read, Update, Delete*). Additionally, some first natural language preprocessing steps are run in this stage.
    - **Similarity Detection module.** Implements all services and domain logic regarding the similarity detection algorithms, including the natural language processes and techniques, as well as the similarity functions and the classification process.

  This architecture is intended to isolate the management of domain data (i.e., requirements data in the Data Management module) with the logic and similarity evaluation of the service *(*i.e.*,* algorithms implementation in Similarity Detection module). The main motivation for this design is to achieve a scalable system which easily allows the integration of new algorithms or techniques. This development will be completely decoupled from the requirements of data management.
- **Requirements DB.** A relational database storing all requirements data.

More details about the software architecture and the technical specifications are provided in section 4.2.

4.1.2. Requirements data: OpenReq schema

The OpenReq project approach is designed to deploy a series of tools or services focused on "*intelligent recommendation and decision system for community-driven requirements engineering"* [7]. As part of this proposal, one of the main features of the designed system is a flexible, scalable environment build upon a microservice

[2] https://swagger.io/tools/swagger-ui/

architecture where each tool or service can be deployed in an isolated workspace, without major dependencies between them.

One of the main requirements for this architecture is to provide a uniform, adaptive interface that allows an easy integration process for any software developer, as well as a comprehensive, standardized communication system that facilitates the usage of these services. This goal is materialized in two software design practices. The first one is related to the RESTful API exposure of each microservice, which is depicted in section 4.3. The second one is related to a shared data communication format between all the tools of the system.

The OpenReq project defines a JSON Schema in order to map the most relevant requirements data inside a unique JSON object that handles all the information related to requirements, projects, stakeholders, and other entities which are common in the RE field and used between the OpenReq microservices. For this reason, this JSON schema is used as the data communication format to use as input and output of the service.

Field	Type	Description	Example
id	string	A unique identifier of the requirement	*"QTCREATORBUG-17803"*
name	string	Short summary or title of the requirement	*"Debugger shows wrong address for pointer treated as array"*
text	string	Long description of the requirement	*"Take a pointer and try to Change the Local Display Format to show it as an Array of 10 items. Instead of dereferencing the pointer and displaying 10 consecutive items at that memory address, [...]"*
project	string	The identifier of the project the requirement belongs to	*"QTCREATORBUG"*
components	list	A list of components to which the requirement refers to	["Debugger"]
type	string	The type of the requirement	*"Bug*"
version	list	A list of versions to which the requirement applies to	[*"Qt Creator 4.2.1"*]
priority	string	The priority level of the requirement	*"NE"* *(Not Evaluated)*

Table 6. Requirements data schema (with examples)

The JSON OpenReq schema is available at Annex A. For simplification purposes, it only includes the relevant part of the schema that the system developed in this thesis is going to use. Table 6 summarizes the schema of the "requirement" data instance, which contains the main fields and attributes used for the similarity detection

algorithms. For further details about how to build a JSON OpenReq requirements dataset, please refer to Annex A and the REST API documentation of the service.

4.2. Software architecture

In this section, the software architecture of the service and its modules are depicted. Additionally, the technical specifications of the tool are enumerated.

4.2.1. Controller-Service-Repository architecture

A basic 3-layer architecture design is proposed, which is composed of controllers, services and repositories. The role of each component is briefly depicted below:

- **Controller.** Defines the specification of the actions (requests) that the service exposes to external systems. It is completely decoupled from the logic of the system, and it is used to deploy the RESTful API (see section 4.3).
- **Service.** Handles all business logic and answers to controller requests. They act as a gateway between the controllers and the repositories (see below).
- **Repository.** Implements access to the database, acting as a gateway layer between the service and the relational database access.

As introduced in Figure 4, the service is designed as two different modules that vertically integrate this architecture. A general depiction of each one of them is provided below.

Data Management module

The Data Management module implements all features regarding the requirements management data operations. All read, delete, write and update operations are processed in this module. Additionally, and as depicted in section 5.2.1, the insert of new requirements is used to apply a basic initial NLP preprocess, which will be used by all the algorithms.

Developers can use this module as a *black box* to handle their requirements data, without requiring further knowledge or implementation modifications.

Similarity Detection module

The Similarity Detection module develops all domain logic handling the evaluation of requirements similarity algorithms. There are two main components: the *BM25FService* and the *FESVMService*. Each one of them implements the features required for each algorithm. Additional classes are designed to extract and decouple some additional features required by each algorithm from the logic of the services.

Figure 5 represents the class diagram of the Data Management Module and the Similarity Module. It includes the integration and the access between these two modules, as well as the application of the 3-layer architecture defined above.

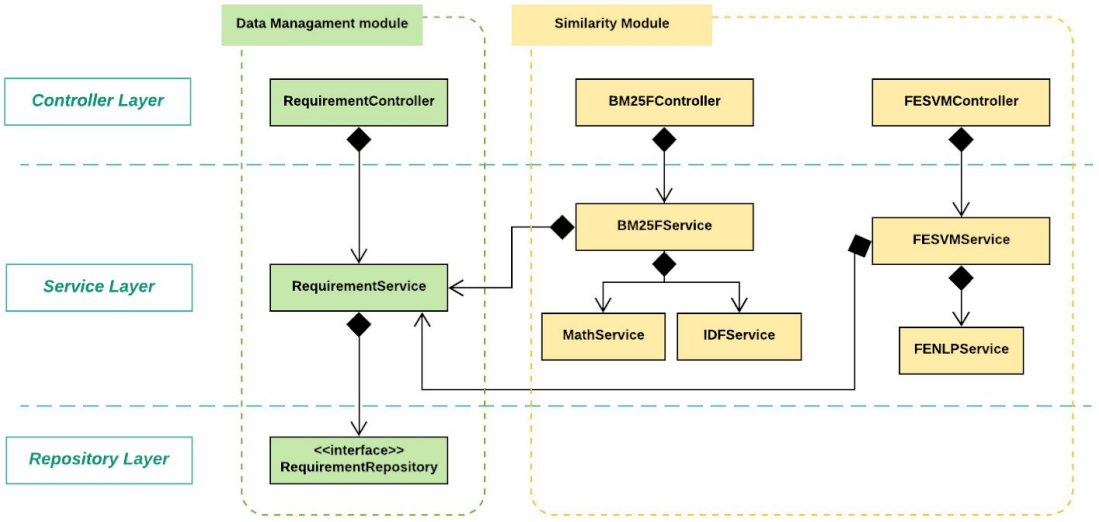

Figure 5. Requirements Similarity modules (class diagram)

Software developers that might be interested in extending the system with new algorithms would only require to do the following steps:

1. Implement a new controller class similar to BM25FController and FESVMController, exposing all available features for the new algorithm.

2. Implement a new service class similar to BM25FService and FESVMService which uses the *RequirementService* to read requirements data. This service class will include all domain-specific logic required by the algorithm.

4.2.2. Technical specifications

Name	Version	Role
Java	1.8	The main programming language used to develop the service [34].
SpringBoot	2.2.0	It allows to create and manage stand-alone web applications with embedded web services [35].
SpringFox	2.9.2	A framework to automatically generate documentation for the API built with Spring [36].
MySQL	8.0.18	A JDBC driver for managing MySQL databases [37].
Apache Lucene	7.4.0	It provides basic NLP open-source techniques like tokenization and stop-word removal [38].
Extended Java WordNet Library	2.0.2	A Java-based API for creating, reading and updating dictionaries in WordNet format. Used for handling synonyms in syntactic features [39].
OpenNLP	1.9.1	It provides basic NLP open-source techniques like advanced sentence boundary disambiguation [40].
StanfordNLP	3.6.0	It provides advanced NLP techniques like a dependency parser [41].
SMILE	1.5.3	It provides basic classification techniques like support vector machine (SVM) classifiers [42].

Table 7. Requirements Similarity technical specifications

Table 7 is a reference list of the technical specifications used in the Requirements Similarity system. Each entry of the list provides the name, the version of the technology, and the role it plays in the domain logic.

Concerning the **quality of the software** and the code of the system, it is proposed to use a code quality analysis tool that automatically detects all issues inside the source code of the project which might not satisfy high-quality standards of software development. For this purpose, the CodeFactor[3] tool is integrated into the Github repository of the Requirements Similarity system.

This tool pulls an automated analysis after each *commit* or change in the project that looks for any kind of issue related to the software quality. These issues include, among others, duplicated code, security vulnerabilities, highly complex methods, coding style, bugs and documentation. After the evaluation, all issues are raised and indexed so they can be consulted using the tool Dashboard. According to the number and the severity of the issues, a grade between F and A is given to each file and to the repository itself.

The Requirements Similarity system is delivered with an **A grade** in all files and in the repository.

4.3. Service integration

As it has been mentioned before, the Requirements Similarity service must be deployed and accessible as an isolated web service via HTTP protocol communication, using a RESTful API that deploys all required features and handles the data representation format proposed in the OpenReq JSON Schema.

This interface is deployed using Spring Boot and Spring Fox. Spring Boot allows software developers to easily deploy stand-alone web applications with embedded web servers. This means that the service can be deployed in any environment as a runnable file without any previous web server configuration. Spring Fox is used to generate a formatted Swagger documentation file which includes all required information for using the API deployed by the service.

[3] Quality report of the repository is available at https://www.codefactor.io/repository/github/quim-motger/tfm

5. BM25F approach: an extension of an Information Retrieval algorithm

The first selected algorithm for requirements similarity evaluation is based on the one described in "*Towards more accurate retrieval of duplicate bug reports*", a proposal by Sun *et al.* [20]. This work is an update of their previous publication "*A discriminative model approach for accurate duplicate bug report retrieval*" [43], where they already addressed similarity detection between requirements in an issue repository.

In this latter work, they migrate from a supervised classification technique using SVM to a new approach in which additional metadata features of the requirements are used to calculate similarity. Between these features, they evaluate an extended version of the BM25F measure, which is the main method used to calculate textual similarity.

5.1. Algorithm analysis

BM25F is a similarity measure from the IR field used for finding relevant documents inside a large dataset of documents or *corpus* given a relatively short text query. This measure is based on two components.

The first one is the *inverse document frequency* (IDF) (see section 3.2.1 for the IDF definition). The second one is the *term frequency* (TF_D), which is the local importance measure of a term *t* inside a document *D*. Consider a document *d* with a set of fields, where each field has its own natural language content. This TF_D measure is defined as follows,

$$TF_D(d,t) = \sum_{f=1}^{K} \frac{w_f * occurrences(d[f], t)}{1 - b_f + \frac{b_f * length_f}{avglength_f}}$$

where w_f is the weight of the field; *occurrences(d[f], t)* is the number of occurrences of *t* in the field *f* of *d; length*$_f$ is the size of the bag of keywords of the field *f* of *d*; *avglength*$_f$ is the average size of the bag of keywords of the *f* field across all documents *D*; and b_f is a parameter *0 <= b <= 1* which applies to the scaling by document length: *b=1* applies to full-length normalization, while *b=0* applies to no length normalization at all.

Based on these two components, the BM25F score of a query *q* (which can be another document or requirement) given a document *d* is defined as follows,

$$BM25F(d,q) = \sum_{t \in d \cap q} IDF(t) * \frac{TF_D(d,t)}{k_1 + TF_D(d,t)}$$

where *t* is each keyword or term shared between *d* and *q,* and k_1 is a tuning parameter to control the effect of the local term frequency.

In the extended approach, the original measure model is extended to better fit long text queries, which is a more suitable approach for the use case of an issue repository, where each query *q* is another bug or issue.

For this purpose, Sun *et al.* propose a $BM25F_{ext}$ approach in which the term frequency of the query *q* (i.e., the issue or requirement to which compare the original issue) is also considered in the formula. Hence, the proposed formula is defined as follows,

$$BM25F_{ext}(d,q) = \sum_{t \in d \cap q} IDF(t) * \frac{TF_D(d,t)}{k_1 + TF_D(d,t)} * \frac{(k_3+1) * TF_Q(q,t)}{k_3 + TF_Q(q,t)}$$

$$TF_Q(q, t) = \sum_{f=1}^{K} w_f * ocurrences(q[f], t)$$

where the tuning parameter k_3 allows controlling the effect of the query term weighting, analogously to k_1.

$BM25F_{ext}$ allows the algorithm to evaluate the textual similarity between textual elements of the issues, like the *title* or the *description.* These natural language attributes can be used to estimate the level of duplicity between two different issues. However, additionally to this measure, Sun *et al.* consider additional metadata like the *project* the issue belongs to, the *component* it is referring to, or even the *version* to which it is addressed to.

This set of features are integrated using an optimized, weighted formula giving a final score between two different documents, i.e., two different issues. This measure is defined as follows,

$$sim(d, q) = \sum_{i=1}^{F} w_i * feature_i$$

where w_i is the weight of each defined feature.

5.2. Process development depiction

Section 5.1 provides a theoretical, mathematical analysis of the $BM25F_{ext}$ information retrieval algorithm and the similarity function proposed by Sun *et al.* In this section, details about the development process of this algorithm are provided, as well as the adaptation to the developed system.

5.2.1. Requirements textual data preprocessing

Before developing the $BM25F_{ext}$ score measure, it is necessary to understand the available data and how it is suitable for this algorithm.

When analyzing the available data of each requirement exported from the Qt repository and the OpenReq JSON schema (see section 4.1.2), it is possible to identify two natural language fields per requirement: *name* and *text*. These fields are easily relatable to the *summary* and *description* fields that Sun *et al.* used for the $BM25F_{ext}$ similarity score evaluation.

Given two requirements R_1 and R_2, and two fields *f = {name, text}*, the first step to evaluate the similarity is to preprocess these natural language fields. For this purpose, a basic natural language pipeline is applied, which is depicted in Figure 6.

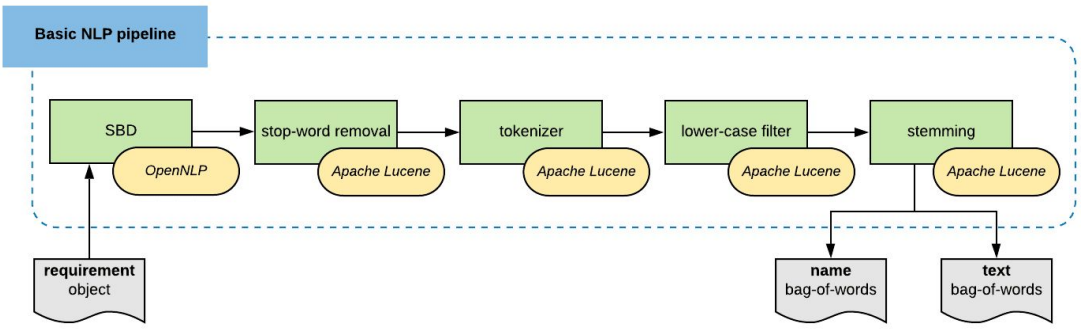

Figure 6. Basic NLP pipeline tasks

As discussed in the FE-SVM approach evaluation (see section 6.1), this basic NLP pipeline is necessary for both algorithm implementations for the Requirements Similarity system. Therefore, a uniform design in which this pipeline is applied when importing requirements to the system is proposed, so that it is not necessary to duplicate work between different algorithms development.

A real requirement from the experimentation dataset defined in chapter 7 is used as an example of the result of applying each one of these techniques. Figure 7 shows each step result of the Basic NLP pipeline for the *name* field, for simplification purposes.

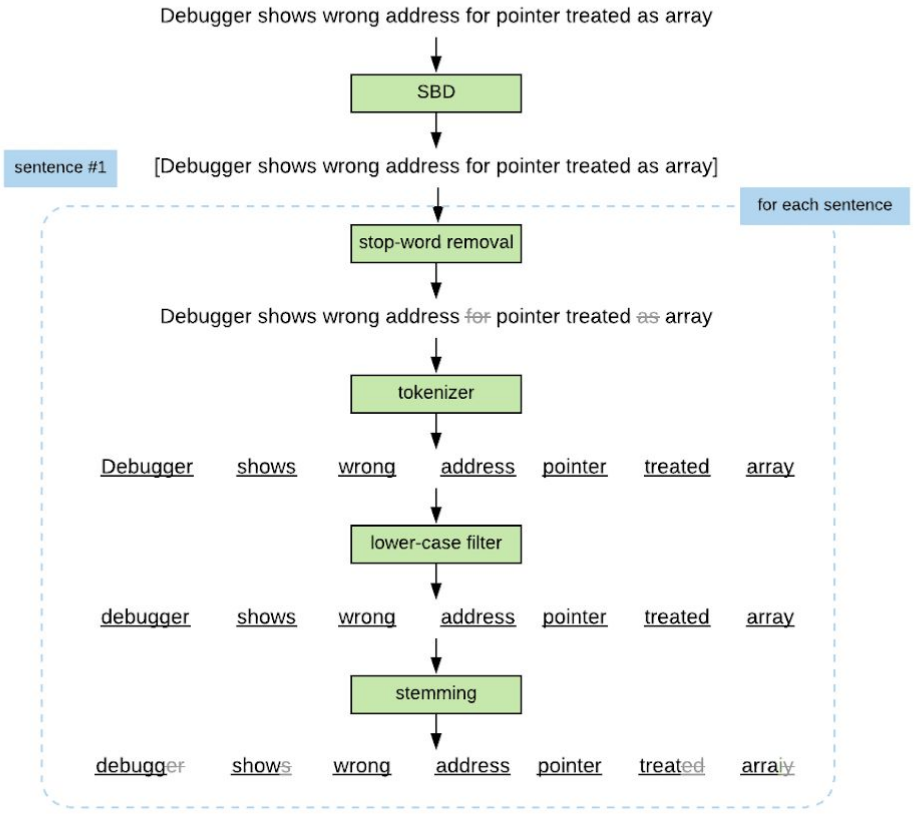

Figure 7. Basic NLP pipeline tasks result - requirement example

The result of each requirement preprocess is two bag-of-words sets, one for each relevant natural language field: *name* (summary) and *text* (description).

5.2.2. BM25F$_{ext}$ algorithm

The next step is to evaluate the *BM25F$_{ext}$* score between R_1 and R_2 for each field *f.*

Sun *et al.* propose to evaluate the *BM25F$_{ext}$* as two different features in the *sim(d, q)* function defined in section 5.1:

- ➔ ***feature$_1$* (Unigram score).** The *BM25F$_{ext}$* score value using requirement *name* and *text* unigrams.
- ➔ ***feature$_1$* (Bigram score).** The *BM25F$_{ext}$* score value using requirement *name* and *text* bigrams.

The default output of each requirement preprocessing pipeline is a bag of words formed by unigrams, where each element is a single term. To evaluate *feature$_2$*, it is necessary to build the bigrams of each field. This is achieved by concatenating in subsets of 2 elements the terms of each field. Notice that for this purpose it is necessary to use the SBD knowledge extracted in the basic NLP pipeline - this way it is avoided to build bigrams between two keywords that do not belong to the same sentence.

5.2.3. Similarity evaluation with metadata integration

In addition to natural language features (i.e., *feature$_1$* and *feature$_2$*) it is required to add to the *sim* function all requirements metadata features that are suitable to improve the similarity score reliability between R_1 and R_2. Based on Sun et al. proposal and in the OpenReq data model, the following features are proposed:

- ➔ ***feature$_3$* (Project score).** Set to 1 if R_1 and R_2 belong to the same Qt project; 0 otherwise.
- ➔ ***feature$_4$* (Type score).** Set to 1 if R_1 and R_2 are of the same type; 0 otherwise
- ➔ ***feature$_5$* (Components score).** Computed by the *Jaccard* similarity between R_1 and R_2 sets of components
- ➔ ***feature$_6$* (Priority score).** Reciprocal distance between the priority of R_1 and R_2 mapped to numerical values

$$feature_6 = \frac{1}{1 + |R_1.priority - R_2.priority|}$$

- ➔ ***feature$_7$* (Versions score).** Reciprocal distance between the latest version of R_1 and R_2 mapped to numerical values

$$feature_7 = \frac{1}{1 + |max(R_1.versions) - max(R_2.versions)|}$$

This approach defines a total number of 7 features that $sim(R_1, R_2)$ computes and weights accordingly to provide a final similarity score for each pair of requirements.

5.2.4. Free parameters optimization

By analyzing the $BM25Fext(R_1, R_2)$ and the $sim(R_1, R_2)$ functions, it can be noticed that this algorithm introduces a total number of 19 tuning parameters, which are listed in Table 8.

These tuning parameters include both the weighting of *features*$_{1\text{-}7}$ and specific term weighting configuration parameters in the $BM25Fext$ formula. The values used for each free parameter to achieve high-quality results are highly tightened to the data and the use case in which the similarity evaluation is applied to. Therefore, it is necessary to apply an optimization process of the algorithms to automatically compute the best values for each free parameter.

Param	Description
w_1	weight of *feature*$_1$ (BM25F unigram)
w_2	weight of *feature*$_2$ (BM25F bigram)
w_3	weight of *feature*$_3$ (project)
w_4	weight of *feature*$_4$ (component)
w_5	weight of feature$_5$ (type)
w_6	weight of feature$_6$ (priority)
w_7	weight of feature$_7$ (version)

Param	Description
$w_{name}^{unigram}$	weight of *name* in *feature*$_1$
$w_{text}^{unigram}$	weight of *text* in *feature*$_1$
$b_{name}^{unigram}$	*b* of *name* in *feature*$_1$
$b_{text}^{unigram}$	*b* of *text* in *feature*$_1$
$k_1^{unigram}$	k_1 in *feature*$_1$
$k_3^{unigram}$	k_3 in *feature*$_1$

Param	Description
w_{name}^{bigram}	weight of *name* in *feature*$_2$
w_{text}^{bigram}	weight of *text* in *feature*$_2$
b_{name}^{bigram}	*b* of *name* in *feature*$_2$
b_{text}^{bigram}	*b* of *text* in *feature*$_2$
k_1^{bigram}	k_1 in *feature*$_2$
k_3^{bigram}	k_3 in *feature*$_2$

Table 8. List of free parameters for the BM25F approach

This optimization process can be summarized in two steps.

The first one is to build a training dataset. For this purpose, it is necessary to have a set of requirement pairs which have been manually classified as duplicates by expert reviewers. Given this set of duplicated requirements as input, Sun et al. propose an algorithm based on building triplets of requirements {R_1, R_2, R_3}, where {R_1, R_2} are instances of the duplicate dataset, and R_3 is a randomly selected requirement from the requirements repository that satisfies that {R_1, R_3} are not members of the duplicate dataset.

With this algorithm, a training set of triplet requirements instances is obtained. In each triplet, R_1 and R_2 are duplicates and R_1 and R_3 are almost certainly not.

To apply an optimization process, it is necessary to define a cost or objective function based on this similarity algorithm. This cost function *C* is defined below.

$$C(\{R_1,\ R_2,\ R_3\}) = log(1 + e^{Y}),\ where\ Y = sim(R_3,\ R_1) - sim(R_2,\ R_1)$$

Lower values of the *C* function imply higher accuracy of the *sim()* function.

For this optimization process, and as suggested by Sun *et al.*, a gradient descent optimization process is applied. For each free parameter, the manual derivative of *C* is manually calculated with respect to each tuning parameter. Next, these free parameters are initialized with default values (i.e., the ones proposed by Sun *et al.*).

Once the optimization process has been set up, a specific number of limited iterations are run. At each loop, a very small variation of one parameter value is applied, which is directly proportional to the partial derivative with respect to that parameter. Thanks to this process, each parameter tends to its optimal value. The number of iterations is used as the stop criteria.

Sun *et al.* provide their own free parameter optimization results, based on public issue repository data. Nevertheless, it is decided to apply a custom optimization process to 7 of the 19 free parameters - specifically to the weight of the features, *i.e.,* w_1 - w_7 parameters. This decision is based on two main reasons:

1. Although there are available results concerning the optimization process of these algorithms, these results could be highly coupled with the dataset used for its validation. This is especially critical with the weight of each feature of the algorithm - the relevance of each attribute or field in the process of detecting duplicates may vary significantly between different datasets. Moreover, it has been necessary to adapt the proposed algorithm to the OpenReq data model to fit differences between some attributes and metadata fields.

2. The analytical and mathematical complexity of manually derive for each free parameter the cost function is very high, especially for those parameters not related to feature weighting. Additionally, when comparing the results obtained by only optimizing the feature weights with the ones reported by Sun *et al.*, it can be concluded that the potential gain of accuracy is not worth the required effort and the schedule deviations risks of dedicating too much time to this task. Therefore, for the rest of the free parameters, it is decided to use those provided by Sun *et al.*

In section 7.3.1 further details about the optimization process, the free parameters values obtained by the optimization step are provided. Additionally, a comparison between the results of the algorithm for different values is depicted.

5.3. BM25F controller

This section briefly introduces the different functionalities deployed by the BM25F controller. These features include all similarity evaluation and experimentation techniques required to both use the similarity detection algorithm and to extract results to evaluate the accuracy of the algorithm.

The BM25F controller exposes the following functionalities:

- **Find top *K* most similar requirements.** Given a specific requirement *R* of a set of requirements and a value *K*, the system computes the similarity score against all other requirements. The result is an ordered list with the top *K* most similar requirements to *R*.

- **Compute similarity between two requirements.** Given two requirements R_1 and R_2, the system returns the similarity score between R_1 and R_2.

- **Compute *recall-rate@k*.** Given a set of requirements, a dataset of duplicate requirement pairs and a *k* value, the service performs an *all-to-all* comparative analysis based on the *recall-rate@k* metric (see section 3.3). The result is the duplicate detection accuracy for each value *i* in the interval *1* <= *i* <= *k*.

- **Free parameter optimization process.** Given a set of requirements and a dataset of duplicate requirement pairs, the optimization process depicted in section 7.3.1 is performed. As a result the free parameters internal values are updated.

For a more detailed, exhaustive depiction of each function and the technical specifications, please refer to the API documentation. It is available through the deployment of the Requirements Similarity service or at the *api/swagger.yaml* file in the Github repository of the project.

6. FE-SVM approach: a feature extraction process with data classification

The second algorithm developed for this master thesis is based on a feature extraction process with a supervised classification step to automatically label pairs of requirements as duplicates or not duplicates.

The development of this approach is mainly based on the proposal by Mahajan and Zaveri [22]. However, the lexical and syntactic features used for this process have been selected across the features studied during the systematic literature review process. The selection process has been carried out accordingly to the most representative and useful features to the defined use case.

6.1. Algorithm analysis

Supervised classification approaches for similarity detection are based on solving the task of classifying a given pair of texts or documents as *D* (duplicates) or *ND* (not-duplicates). Mahajan and Zaveri focus on applying this technique to evaluate sentence paraphrasing. Therefore, this section only refers to similarity at a sentence level. Section 6.2 depicts the details and the adaptation process to requirements similarity.

This duplicate detection system is based on submodules or steps. These modules are defined as Mahajan and Zaveri:

1. **Preprocessing module.** Given two input sentences A and B, the first step is to apply an NLP pipeline process to the natural language text, in order to transform this natural language to a bag of words or tokens without noise or ambiguous tokens to use as input for the Feature Extraction module. This pipeline includes the following techniques: tokenization, stop-word removal, POS tagging, lemmatization, and dependency parsing.

2. **Feature Extraction (FE) module.** The preprocessing module produces two different outputs: the first one is the bag of words or tokens for each sentence; the second one is a dependency tree for each sentence. The bag of words of A and B are used to extract lexical features for the A-B pair. These lexical features include matching between tokens or keywords between the two sentences, independently of the role these tokens play in each sentence.

   On the other hand, the dependency tree is used to extract syntactic features. Each term or node of the dependency tree is labelled with a specific syntactic role. Based on this identification and the relations between these terms, the syntactic features are computed by looking for grammar patterns shared between sentences A and B. These include, for instance, two sentences sharing the same subject or two sentences whose predicate share the same direct object.

Section 6.2.2 provides further details about the lexical and syntactic features selected and developed for the feature extraction process.

3. **Classification module.** Mahajan and Zaveri propose a classification module that implements a set of classifiers which include: SVM, Naïve Bayes, Voted Perceptron and Multilayer Perceptron. They use these classifiers to provide different predictions for the classification of the A-B pair. Finally, they use a voting system (i.e., the most frequent predicted label) to decide the system prediction.

There are two critical decisions to consider in the development of this algorithm. The first one is the set of features to extract during the FE process. For this purpose, a total number of 14 features have been chosen (see section 6.2.2) which are the most frequent features used for this kind of approaches between the systematic literature review results.

The second one is the type of classifier to use for the supervised classification process. By taking a look at the contributions analyzed in the literature review process, there seems to be a general consensus in the usage of SVM classifiers for natural language similarity detection, including the analysis of Mahajan and Zaveri. Therefore, it is decided to exclude the classifier selection process of the scope of this thesis.

6.2. Process development depiction

This section describes the development process and the implementation of the FE-SVM algorithm. In this section, the algorithm is applied to the requirements data domain, and further details about the specific features and the classification process are provided.

6.2.1. Requirements textual data preprocessing

The NLP pipeline for the natural language fields preprocessing of the requirements is depicted in figure 8.

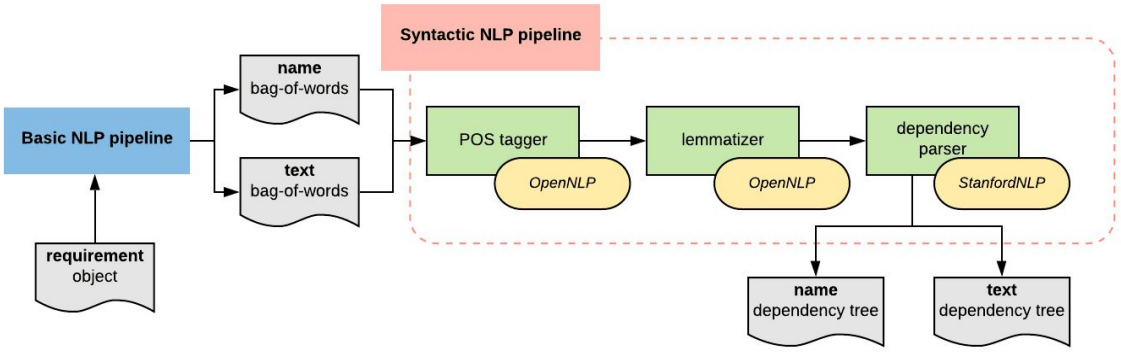

Figure 8. Syntactic NLP pipeline tasks[4]

[4] Basic NLP pipeline depicted in Figure 6

As mentioned in section 5.2.1, and for design and efficiency reasons, the basic, lexical natural language preprocessing output is exploited in the syntactic natural language process necessary for the syntactic feature set. The steps executed during this NLP steps are necessary for the subsequent steps of the syntactic NLP pipeline, except for the stemming step. If stemming was applied before the POS tagger step, the tags would not be based on the original words, and therefore results would be compromised. Additionally, this Syntactic NLP pipeline handles lemmatization after the POS tagging, which is a deeper step than stemming itself.

The output of this pipeline is a dependency tree for each sentence inside the *name* and the *text* fields of each requirement. Notice that this is one of the first differences with respect to the Mahajan and Zaveri proposal. While the originally described algorithm evaluates paraphrasing at a sentence level, requirements *name* and *text* fields are not limited to be single-sentence texts. Therefore, it is required to handle a dependency tree for each sentence in each field.

The same requirement *name* field as in Figure 7 is used as an example to demonstrate how the syntactic preprocess is handled by the developed system. The results of this Syntactic NLP pipeline are shown in Figure 9.

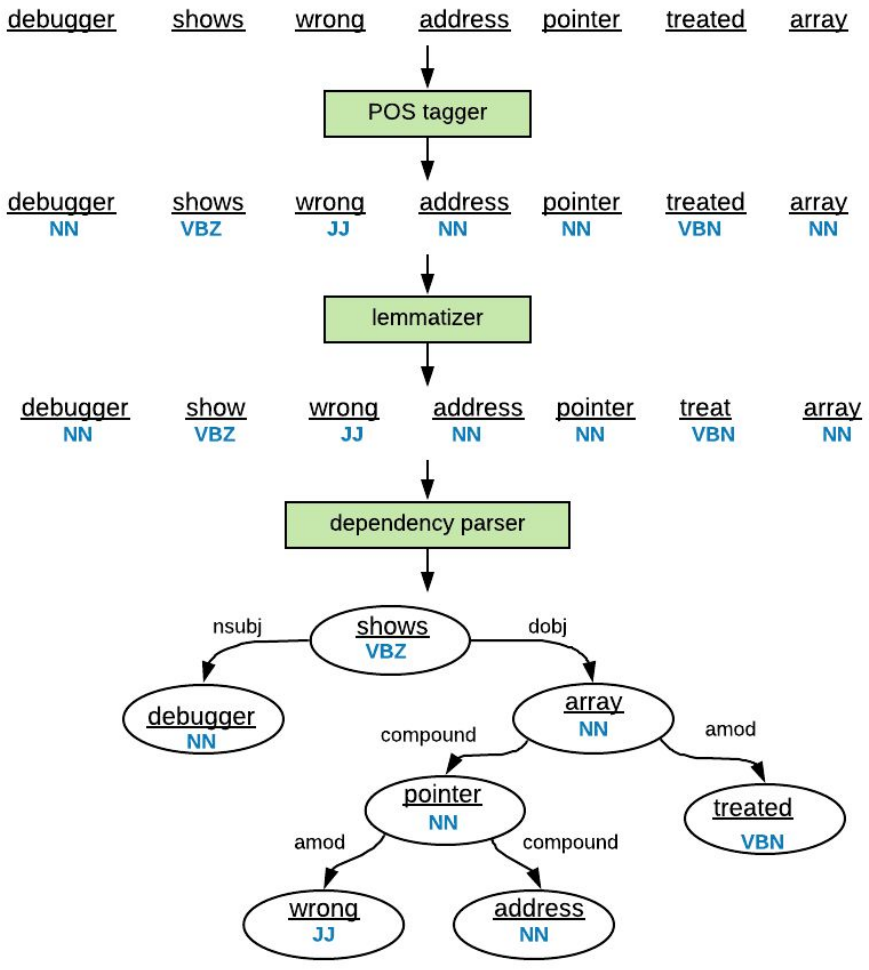

Figure 9. Syntactic NLP pipeline tasks result - requirement example

Each dependency tree node provides the information of the lemma and the POS tag assigned to each token in the original *bag-of-words* input of the pipeline. This information is necessary for the feature extraction process defined in the next section.

6.2.2. Feature Extraction process

The first step of this stage is to define which features are going to be extracted using an align-based approach to define each requirement pair instance. A set of 14 features is proposed: 6 lexical features and 8 syntactic features.

Given a pair of requirements R_1 and R_2, each feature must be defined as a numerical value or score based on the level of alignment between R_1 and R_2 according to that feature.

Notice that if the implementation was based on the original algorithm described by Mahajan and Zaveri, it is not only important to adapt these features to generic texts with multiple sentences. The requirements data is composed of two natural language fields: *name* and *text*. Therefore, each one of the features proposed by this and other sentence-similarity contributions must be translated into 2 features: one for the *name* field and another for the *text* field.

6.2.2.1. Lexical features

The 6 lexical features proposed by the FE-SVM algorithm are introduced below. Notice that each feature is identified by an *i-i+1* subindex, as each feature must separate the *name* attribute score from the *text* attribute score for the feature extraction process.

- ➔ ***feature*$_{1\text{-}2}$ (Word overlap).** Computes the ratio of overlapping words or tokens between the R_1 and R_2 original texts. This intersection is expanded using synonyms found using the WordNet synonym dictionary. The score is computed using the Jaccard similarity function.

- ➔ ***feature*$_{3\text{-}4}$ (Unigram match).** Computes the Jaccard similarity between the *bag-of-words* sets of R_1 and R_2.

- ➔ ***feature*$_{5\text{-}6}$ (Bigram match).** Analogue to *F2,* but using bigrams as the elements to match between R_1 and R_2 *bag-of-words* sets. Similar to the bigram extraction process in the BM25F approach, sentence boundary information is used to avoid creating false bigrams belonging to different sentences.

Notice that this set of features does not require any of the output information generated by the Syntactic NLP pipeline described in Figure 8. This means that, when evaluating the results of the FE-SVM using only lexical features, it is not necessary to run the Syntactic NLP pipeline. This is significantly important because the dependency parser process is time-consuming. Therefore, the algorithm must avoid executing unnecessary work for efficiency purposes.

6.2.2.2. Syntactic features

The 8 syntactic features proposed for the FE-SVM algorithm are introduced below. The same criterion is used for the enumeration: each feature is enumerated for the *name* attribute and the *text* attribute.

- ➔ ***feature*$_{7\text{-}8}$ (Subject match).** For each requirement in the pair, the subject of each sentence is extracted: i.e., the algorithm looks for the node in the dependency tree whose tag matches the **sub** regexp pattern. A set of subjects is obtained for R_1 and another one for R_2. To these sets, the Jaccard similarity is applied to get a match score.

- ➔ ***feature*$_{9\text{-}10}$** (**Subject-verb match).** The algorithm looks in each sentence for all grammar patterns matching a dependency relation between a *subject* and a *verb,* and the dependency label is kept. For all instances found in R_1 and R_2, the Jaccard similarity is applied to get a match score, where a match is found when two instances share the same subject, the same verb and the dependency label value.

- ➔ ***feature*$_{11\text{-}12}$ (Object-verb match).** Similar to F5, but the governor of the relationship must be an *object* instead of a *subject*. Jaccard similarity score and matching criteria are computed analogously.

- ➔ ***feature*$_{13\text{-}14}$ (Noun match).** For all *nn* (i.e., compound nouns) dependency relationships found in each text field, the Jaccard similarity is applied to the set of node pairs joined by this relationship.

These features require the dependency parsing process defined in the Syntactic NLP pipeline, which includes POS tagger, lemmatization and the dependency parsing itself. The first two techniques are required to correctly build the dependency tree, so each node is identified by the original term, the lemma, and the POS tag assigned to it.

The result of each feature extraction process is a set of requirement pairs {R_1, R_2} for each instance of the collection with a total number of 14 features. As mentioned before, for experimentation purposes, the FE-SVM tool must provide functionalities to apply only a specific set of features.

6.2.3. Support-Vector-Machine classification

The dataset of featured instances extracted from the FE process is ready to be sent to the classification module. This classification process is represented by two main tasks:

1. **Training.** The classifier receives a dataset of labelled requirement pairs, where each pair is labelled as *D* (duplicate) or *ND* (not-duplicate). This data is used to train the model of the classifier, which will be used for future requirement pair predictions.

2. **Prediction.** Using an already trained classification model, new not-labelled requirement pairs are sent to obtain a class prediction. This prediction will be

based on projecting the featured requirement pair in the multidimensional model space and identifying in which part of the model (i.e., which class) it is placed.

Previous tasks are a simplification of typical classification processes applied to the use case. Section 6.3 develops the features exposed by the developed system regarding testing, optimization and evaluation purposes.

The literature review has proven that SVM is the most commonly used classifier in NLP feature extraction processes with paraphrase detection based on supervised classification techniques. This master thesis focuses on developing and optimizing the configuration of this classifier.

6.2.3.1. Classifier configuration optimization

SVM classifiers must be tested and optimized with the most appropriate configuration to achieve the best possible results given a specific domain or dataset. For this purpose, during the experimentation stage, it is necessary to use the labelled data of *D* and *ND* pairs of requirements to look for the best parameter configuration. Details about the experimentation process for this purpose are detailed in chapter 7.

There are 3 configuration parameters which need to be evaluated:

1. **Kernel function.** The set of mathematical and analytical functions that are used during an SVM classification process to find the optimal hyperplane equation to separate the classification data is defined as the kernel function. The most basic and easy to understand for binary classification tasks is the linear kernel. However, usually, datasets are not linearly separable, and therefore this kernel function is not the most suitable.

   Although the features seem to be linearly separable, it is not possible to be sure without deep data analysis. For this purpose, two different kernel functions are evaluated: **linear** and **RBF** (Gaussian radial basis function).

2. ***C* parameter.** The *C* parameter is a configuration variable that states which level of misclassifications are allowed by the hyperplane defined by the kernel function.

   Large values of *C* imply a very small margin between the hyperplane and the nearest data instances (i.e., the nearest requirement pairs), which means that most of the instances are correctly classified, as the hyperplane is very coupled to the distribution of the training dataset.

   Low values of *C* imply a wider margin between the hyperplane and the nearest data instances, which means that some instances might be wrongly classified, but the hyperplane is a more generic solution.

3. ***sigma (γ)* parameter.** It is only used in the RBF kernel. This parameter affects the classification process in a similar way that *C* does. It configures the sensitivity on the hyperplane definition. Higher values imply an overfitted

approach to the training data, with low misclassifications. Lower values imply a more generic approach, but with more misclassified requirement pairs.

The details and results of this optimization process are described in section 7.4.1.

6.3. FESVM Controller

Once the algorithm is fully developed, it is necessary to design the features and specific actions that will be exposed as REST API requests for software developers to use and test the system. Similar to the BM25F approach, it is necessary to design these features based on providing a useful similarity detection system and on defining experimentation and evaluation tools for study purposes.

These features are defined as follows:

- **Train classifier.** Given a set of requirements and a set of labelled requirement pairs, the system builds and trains an SVM classifier. This dataset of labelled pairs must be significantly balanced to provide minimum quality results, so a minimum representation of both duplicate and not-duplicate pairs is necessary to avoid overfitting one of the classes.

- **Test classifier.** The previously trained classifier can be used to predict new requirement pairs classifications. For each pair, the algorithm applies the feature extraction process. After this step, it sends this data to the SVM classifier and provides a class prediction, returning this result to the user.

- **Train and test (cross-validation).** Given a set of requirements, a set of labelled requirement pairs, and a *k* value, the system performs a *k*-cross-validation process with the dataset of labelled requirement pairs.

  The result is the aggregated confusion matrix of this cross-validation. Notice that it has been decided to provide the original output data (i.e., TP, TN, FP, FN values) instead of already computed metrics like the accuracy, the precision or the recall of the validation process. This approach allows software developers to compute their own quality metrics, according to their needs.

- **Configuration optimization.** Functionalities to calculate the most optimal SVM configuration are provided. This feature is run as a cross-validation process, but with additional input data: the kernel to be evaluated (LINEAR or RBF); the *C* values to be tested; and the *sigma* values to be tested.

All features have additional parameters which allow developers to decide whether to apply lexical features, syntactic features or both set of features to each process. By default, all sets of features are used. Additionally, all features allow SVM configuration by request (i.e., kernel, *C* value and *sigma* value). If they are not sent, the system uses default parameters, which are stated based on the experimentation results.

For a more detailed, exhaustive depiction of each function and the technical specifications, please refer to the API documentation available in the repository.

7. Empirical experimentation: Qt's use case

This chapter provides an analysis of the quality results of the Requirements Similarity system. For this purpose, and as it has been referred to across this document, a real use case validation is applied using one of the OpenReq partners scenarios and data to study the accuracy and the performance of the algorithms, as well as to compare and conclude which is the most suitable solution for this use case.

This chapter starts by providing a general overview of Qt's scenario. After the use case requirements have been set, the experimentation details are developed: the configuration, the features of the system to be used, and the obtained results.

7.1. Use case description: duplicate detection in an issue repository

The Qt Company is an international software development company which focuses on providing cross-platform software frameworks for the development of apps and devices [45]. Their main goal is to provide integrative, easy-to-use environments for users and developers to build and develop their own applications. This goes from desktop and embedded end-user applications to business-critical processes.

As part of their business vision based on providing open-source solutions, they additionally work on creating and maintaining a global network of users and developers working and using their products. Between some of these tasks, they provide a **public issue repository** based on JIRA. This bug tracking system allows users to report, discuss and work on issues reported by end-users across the different projects that the Qt company manages.

As one of the main problems identified in RE, managing and identifying duplicated issues reported by different users is a time-consuming problem. This duplicated requirement detection process can be solved by the solution developed in the Requirements Similarity system.

The developed system must be capable of handling responsive requests regarding the evaluation of duplicated requirements. This means that, in addition to reliable and accurate results, the system must be able to run comparisons between a new requirement and a set of already existing requirements almost instantaneously. Consequently, performance and efficiency are important dimensions to evaluate from the developed solution.

Qt's available data in their public issue repository consists of a large dataset of requirements with more than 110,000 reported issues across 20 projects or software products.

7.2. Technical experimentation preparation

Once the scenario has been defined, it is necessary to import and build the required data for validation purposes. This data is based on two different kinds of information that the developed system requires.

The first one is the requirements dataset itself. These are imported using the REST API of their web service (see section 4.1.1). The second one is the duplicate/not-duplicate dataset of requirement pairs. BM25F optimization process and FESVM model training process both require labelled information, i.e., a set of requirement pairs validated as duplicates and another set labelled as not duplicates.

The set of duplicate pairs has been exported from the Qt's JIRA issue repository. This system allows the manual identification of duplicated reported issues, marking a specific requirement as a duplicate of another one. For this purpose, it was necessary to look for issues with an existing duplicate relation. The set of not duplicate pairs has been built randomly between non-related requirements with a later review by an expert.

Table 9 summarizes the details about the available experimentation data.

# total requirements				111,143	
# projects				20	
# requirements per project					
QTSOLBUG	179	**QTWEBSITE**	657	**QTSYSADM**	246
QTBUG	72,976	**QTCOMPONENTS**	1,065	**QTPLAYGROUND**	15
QTVSADDINBUG	620	**QTIFW**	1,275	**AUTOSUITE**	1,145
QTCREATORBUG	20,380	**QDS**	1,011	**QTWB**	26
QTJIRA	270	**QBS**	1,378	**QT3DS**	3,532
QTQAINFRA	2,813	**COIN**	378	**QSR**	480
QTMOBILITY	1,781	**PYSIDE**	916	**-**	-
# labelled duplicates				1,436	
# labelled not-duplicates				1,499	
# training set instances				2,935	

Table 9. Summary of experimentation data

All requirements data is imported to the Requirements Similarity system using the Data Management module. Once this process has finished, the system is ready to begin with the optimization and evaluation experiments.

7.3. BM25F experiments

The experimentation stage of the BM25F algorithmic approach is classified in 3 steps. First of all, it is necessary to apply the optimization process to the tuning parameters using the dataset of labelled duplicates. Second of all, a general quality evaluation measure with the *recall-rate@k* metric is applied, which will allow us to compare the developed solution with the results reported by the original publication this development has been based on. Finally, an evaluation technique is introduced to compute results data which can be used to be compared with the FE-SVM approach.

7.3.1. Free parameters optimization process

Using the set of 1,436 labelled duplicated requirements, a triplet training set of requirements is built. These requirements are used to optimize the weights of $features_{1\text{-}7}$ as described in section 5.2.4. For each tuning parameter, Table 10 reports the initial value, the reported value by Sun *et al.*, and the optimized value obtained by the algorithm using this training data.

param	*initial*	*reported*	*optimized*
w_1	0.9	1.163	0.352
w_2	0.2	0.013	0.004
w_3	2.0	2.285	2.049
w_4	0.0	0.032	0.047
w_5	0.7	0.772	0.970
w_6	0.0	0.381	0.012
w_7	0.0	2.427	0.111

Table 10. Free parameter optimization results

There are significant differences among some of the parameters, especially between the weights of $feature_1$ (BM25F with unigrams) and $feature_2$ (BM25F with bigrams). The most logical explanation for this phenomenon is the natural language data of Qt's dataset (i.e., how the requirements are written there). The score provided by the BM25F algorithm is directly proportional to the length of the natural language texts. Therefore, a corpus with larger documents will provide greater scores for these features. Smaller values for w_1 and w_2 probably imply that, from a global perspective, documents in Qt's dataset are larger than documents in the datasets used by Sun *et al.*

Furthermore, when comparing the results, it can be observed that the relation between w_1 and w_2 is almost identical. The reported w_1 value is ~4 times greater than the optimized w_1 value. The same proportion is applied to w_2. Therefore, the proportional weights of both features are very similar, although their absolute values are not.

7.3.2. Quality evaluation: *recall-rate@k*

As a first evaluation of the BM25F approach, a *recall-rate@k* test is run. As reported by Sun *et al.*, this is a very high time-consuming task, which can take days of execution time to be finished in large project datasets (i.e., projects with tens of thousands of requirements). However, it is important to run this evaluation to compare the results with the ones reported by Sun *et al.* before this algorithm can be compared with the

FE-SVM approach. Hence, a *recall-rate@k* experiment is run with the details depicted in Table 11.

K-value	20
Projects	QTQAINFRA PYSIDE QTIFW
Nº requirements	5,004
Nº duplicates	57

Table 11. Recall-rate@k experiment set-up

This experiment is repeated with 3 sets of parameter configurations: the 1st experiment is run with the initial parameter values, without any optimization process; the 2nd experiment is run with the parameters reported by Sun *et al.* in their publication; the 3rd experiment is run with the optimization parameter values.

Figure 10 shows the *recall-rate@20* results. By comparing the results after the optimization process with the ones obtained without any optimization or with the values of the tuning parameters reported by Sun *et al.*, it is possible to observe a clear improvement that demonstrates the importance of the optimization process. This improvement is translated into *K* points concerning the number of duplicates found.

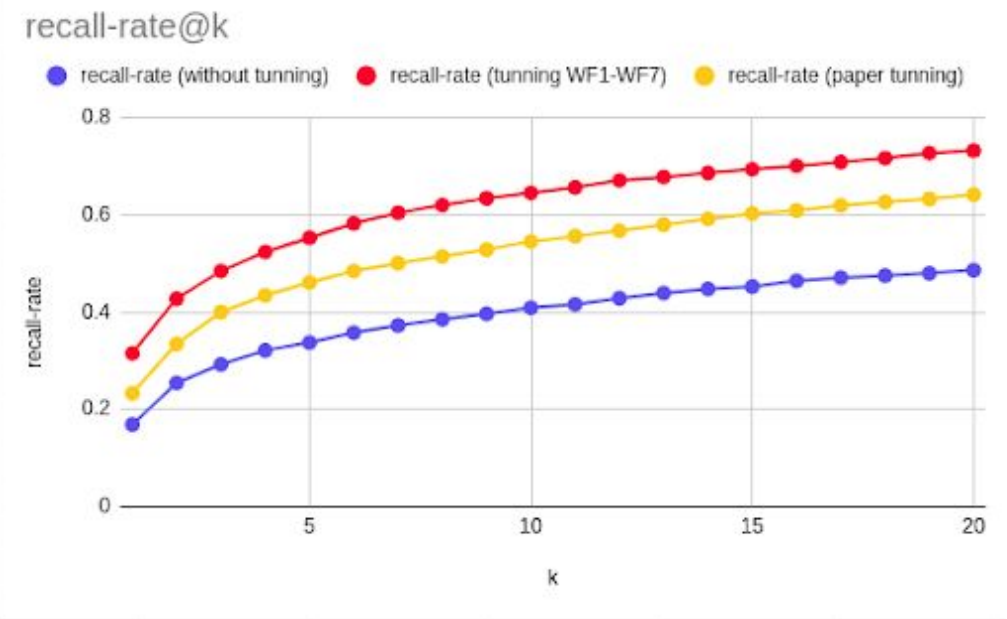

Figure 10. Recall-rate@20 experiment results

Additionally, if the *recall-rate@20* analysis is compared with the results reported by Sun *et al.*, very similar results are observed. They do not provide quantitative data, but the plots reported in their publication estimate a ~0.70 value at *k=20*, which is very similar to the results reported in Figure 10.

7.3.3. Duplicate discernment: threshold evaluation

After a quality evaluation based on state-of-the-art comparison with up-to-date developments of the BM25F algorithm approach, it is necessary to focus on evaluation in an analytical comparison between both developed approaches.

The main challenge to provide comparative results is adapting two different solutions to a unique, uniform format in which to use quality metrics like the ones described in section 3.3. BM25F approach is based on solving the situation in which, given a populated issue repository, a new requirement entry is compared against all existing requirements to provide suiting duplicate candidates of the new entry. In this approach, there is no specific classification process in which to discern between duplicate and not duplicate pairs of requirements. On the other hand, the FE-SVM approach is a typical supervised classification problem in which each pair of requirements is evaluated as a duplicate candidate, and a specific category (*D* or *ND*) is provided.

In order to compare the accuracy and other quality metrics of both algorithms, it is proposed to use a *threshold* evaluation technique for the BM25F solution. By estimating a threshold score to the $sim(R_1, R_2)$ function of this approach, data can be split between two classes (*D* or *ND*) based on the score given to a pair of requirements.

The main task of this challenge is identifying the optimal score or *threshold* to consider a pair of requirements as duplicates. However, this task can be achieved using the duplicate and not-duplicate sets of pairs. This set of merged requirement pairs (~3000 instances) is sent to the BM25F algorithm and a score for each pair is obtained. With this information, a cross-validation process is applied using the BM25F score as the only feature of the dimension.

Table 12 summarizes the best threshold results, the confusion matrix and the accuracy metrics used for evaluation.

threshold	TP	TN	FP	FN	Accuracy	Precision	Recall	F-measure
3.50	1297	1434	44	100	94.99%	96.72%	92.84%	94.74%

Table 12. Cross-validation results (BM25F) with threshold

7.4. FE-SVM experiments

In the second part of the experimentation stage, the FE-SVM approach is optimized and the reliability results are analyzed. The main difference concerning the previous scenario is the lack of additional work to provide quality results. As being part of the supervised classification solution family, cross-validation provides us with the confusion matrix and the quality metrics of the classification process.

7.4.1. SVM classifier optimization process

As introduced in section 6.3, a feature in the FE-SVM controller has been defined which handles an optimization process based on cross-validation with 3 sets of parameters: the kernel function, the *C* parameter, and the *sigma* parameter (only used for RBF kernel). Based on the literature review, it is proposed to apply cross-validation

to all combinations between the values described in Table 13. Notice that, for the linear kernel, the sigma value is not required.

param	values	param	values	param	values
kernel	*LINEAR* *RBF*	*C*	0.001 0.01 0.1 1 10 100 1000	*sigma*	0.001 0.01 0.1 1 10 100 1000

Table 13. SVM configuration optimization parameters

A 10-cross validation analysis is run to the merged datasets of duplicate and not-duplicate pairs of requirements (~3000 requirement pairs). For simplification purposes, this optimization process is applied with both syntactic and lexical features, leaving the analysis of these set of features to the quality evaluation analysis once the SVM classifier has been configured optimally.

Annex B expands all results for each possible configuration of the classifier. This includes a total number of 7 configurations for the linear kernel and 49 configurations for the Gaussian RBF kernel. Table 14 summarizes the best configuration results.

Kernel				RBF			
***C* value**				1			
***sigma* value**				0.01			
TP	**TN**	**FP**	**FN**	**Accuracy**	**Precision**	**Recall**	**F-measure**
1303	1242	249	116	87,46%	83,96%	91,83%	87,71%

Table 14. SVM configuration optimization results (summary)

As stated in the table above, the RBF kernel gives better results in the classification process that the linear one. However, by taking a look at Annex B results, it can be observed that the maximum accuracy reached by the linear kernel is only 3 points lower than the best result with the RBF kernel. This means that the dataset is linearly separable, but an RBF kernel is more suitable to optimize the results of the defined scenario. The other parameters are suited accordingly to the maximum accuracy.

7.4.2. Quality evaluation: lexical & syntactic cross-validation

The optimal configuration reported in the optimization process is used to apply cross-validation with *k=10* and to compare 3 different scenarios using the FE-SVM approach. These scenarios relate to the usage of only lexical features, only syntactic features, or both sets of features. The datasets are the same ones used in the previous optimization process. Table 15 summarizes these results.

	TP	TN	FP	FN	Accuracy	Precision	Recall	F-measure
Lexical	1284	1322	169	135	89,55%	88,37%	90,49%	89,42%
Syntactic	623	1389	102	796	69,14%	85,93%	43,90%	58,12%
both	1303	1242	249	116	87,46%	83,96%	91,83%	87,71%

Table 15. Cross-validation results (FE-SVM) with *k=10*

Based on these results, the maximum accuracy is achieved when using only the lexical set of features. The usage of only syntactic features provides very poor results. In fact, a very low recall is obtained when using only syntactic features. This means that a lot of real duplicate instances are wrongly labelled as ND.

The most logical explanation to this phenomenon is that considering only grammatical structures when identifying duplicates between requirements is not the best approach. Different circumstances like the writing style of a user may affect the capacity of the algorithm of comparing grammatical structures and finding a resemblance between actual duplicated requirements.

Furthermore, if lexical and syntactic features are combined, the results are highly improved, but not as much as in the scenario with only lexical features. From these metrics, it can be concluded that syntactic features do not seem to contribute to improving the FE-SVM approach, as some contributions in the literature review suggested. Therefore, these are discarded from the algorithm evaluation.

7.5. Comparative evaluation between algorithms

After the experiments have been run, it is time to apply a comparative, qualitative analysis between the BM25F-based approach and the FE-SVM approach. This analysis is based on two dimensions. The first one is related to the accuracy and the quality of the results provided by each algorithm, which has been the focus of this thesis. The second one is based on the performance and, to be specific, the execution time required by each algorithm to perform the same processes.

7.5.1. Accuracy and solution quality

As depicted in section 3.3, 4 quality indicators are used for each algorithm: accuracy, precision, recall, and f-measure. In this section, the results obtained by each algorithm are compared.

For the BM25F algorithm, the optimization values of the free parameters depicted in Table 10 are used as the default configuration. For the FE-SVM algorithm, the kernel and the configuration values defined in Table 14 are used to run the crossover validation.

	Accuracy	Precision	Recall	F-measure
BM25F	94.13%	96.27%	91.64%	93.90%
FE-SVM	89,55%	88,37%	90,49%	89,42%

Table 16. Algorithm qualitative results comparative analysis

As it can be easily analyzed from table 16, the BM25F approach provides the best results in all the metrics. The accuracy, which estimates the number of correctly identified pairs (whether they are duplicates or not duplicates) is almost 5 points higher in the BM25F approach.

However, a significantly higher distance is observed between the precision metrics of both algorithms (>8 points) rather than in the recall metrics (~2.5 points). This means that the BM25F approach is especially better than the FE-SVM approach in avoiding misclassifying not-duplicates as duplicates. On the other hand, the difference is lower when focusing on how well the algorithms identify all existing duplicates in a dataset.

The *f*-measure metric provides the harmonic mean between the precision and recall values. In this case, as both input values are higher in the BM25F approach, the *f-measure* is also better.

Despite the fact that the BM25F approach provides better results when using a threshold evaluation, it is important to remark the high-quality results obtained by the FE-SVM approach. Both accuracy and f-measure are very close to a 90% value, which are very good qualitative results.

The differences between the approaches are not that high, and the most appropriate conclusion would be that, even though for this scenario the BM25F seems to provide more accurate results, both solutions should be tested in any new scenario. The nature of the dataset, such as the length of natural language fields or the type of users introducing new issues, might change what is the best option for duplicate requirements detection.

7.5.2. Performance: execution time

Although the literature review does not provide a lot of information regarding the performance and efficiency of the algorithms, it is important to evaluate the execution time required to perform the main actions from this scenario.

An execution time evaluation based on a horizontal and vertical analysis is proposed. From the vertical perspective, it is necessary to evaluate and compare the total amount of time required by each algorithm to complete the same use case action. From the horizontal perspective, it is required to evaluate and compare the different steps of each algorithm one by one, in order to understand which tasks are the most time-consuming. To better understand this proposal, this process evaluation is depicted in Figure 11, which provides a generic abstraction of the main tasks involved in the duplicated requirements detection scenario for each algorithm.

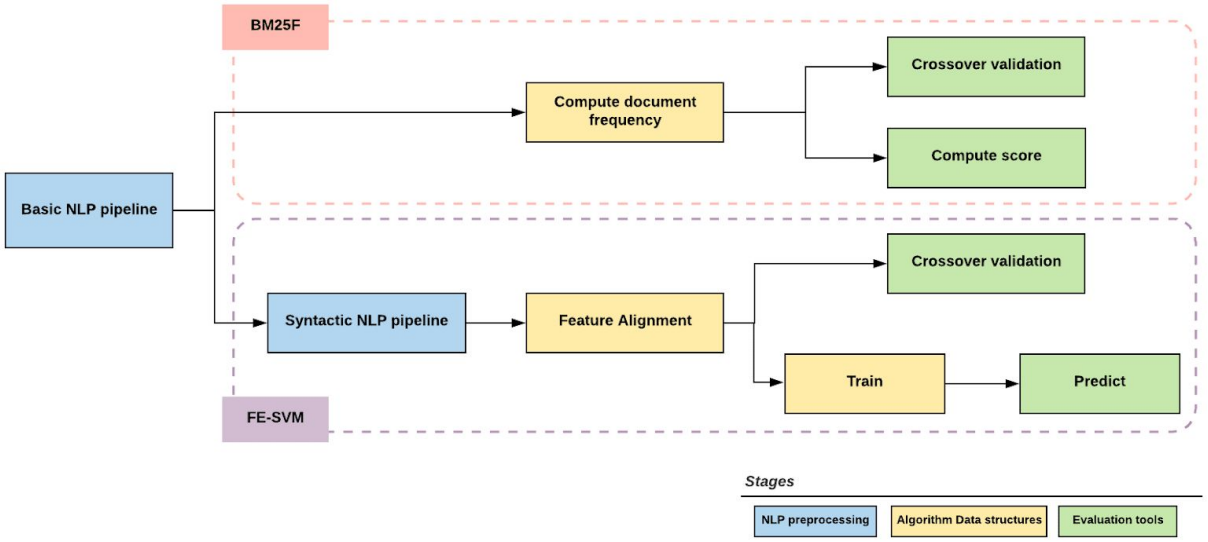

Figure 11. Requirements Similarity system tasks (generic representation)

Each path or branch of the tree defines a specific action. Cross-validation paths define the process of using all duplicate and not-duplicate dataset instances used during the experimentation stage. The '*Compute score*' and *'Train+Predict'* branches are the specific features of each algorithm to evaluate 1 pair of requirements.

The next step is to focus on evaluating and measuring the different intrinsic steps of each similarity detection technique. For this purpose, those tasks that are not related directly to the algorithm itself but to the software architecture design or the used technologies are ignored. These include, for instance, read/write DB operations.

Each experiment is run 5 times in an Intel Core i7 (8th generation) with 16GB RAM, and the average execution time of these 5 executions is used. The results are summarized in Table 17.

Some important considerations are raised by analyzing vertically and horizontally the execution time results. These are summarized as follows:

1. By analyzing from scratch each algorithm process (i.e., including pre-evaluation and evaluation stages), it can be observed that the approach requiring the less amount of time to preprocess requirements data and provide an evaluation between 2 requirements is the BM25F approach (~26 seconds).

   However, this observation neglects a very important feature of the BM25F approach. A compute score evaluation without a previous cross-validation does not provide us with a *threshold* score that allows us to evaluate whether a requirement is a duplicate or not. Furthermore, if the algorithm is replicated as suggested by the original publication, a single requirement-to-requirement comparison does not provide useful information, as the potential of this approach is to compare results with other requirements and to obtain the most similar existing requirements of an existing one.

2. In the FE-SVM algorithm, using syntactic analysis requires a lot of execution time, while as demonstrated in the previous section it does not improve quality results. Both observations are a clear indicator that using syntactic data in this algorithm is not recommended.

3. The FE-SVM approach requires more time to apply the pre-evaluation stage but provides very good results during the cross-validation step (~6 seconds for almost 3,000 requirement pairs) and during the prediction step (<1 ms). The prediction task (which only includes the feature alignment between 2 requirements and the prediction by the SVM) is extremely efficient.

<table>
<tr><td>Algorithm</td><td colspan="2">BM25F</td><td colspan="4">FE-SVM</td></tr>
<tr><td>Stage</td><td colspan="6">Action</td></tr>
<tr><td rowspan="3">NLP preprocessing</td><td colspan="6">Basic NLP pipeline</td></tr>
<tr><td colspan="6">20,886 ms</td></tr>
<tr><td colspan="2" rowspan="2">-</td><td colspan="2">With Syntactic NLP pipeline</td><td colspan="2">Without Syntactic NLP pipeline</td></tr>
<tr><td colspan="2">132,009 ms</td><td colspan="2">0 ms</td></tr>
<tr><td rowspan="4">Algorithm Data Structures</td><td colspan="2" rowspan="2">Compute document frequency</td><td colspan="2">Feature Alignment</td><td colspan="2">Feature Alignment</td></tr>
<tr><td colspan="2">23,624 ms</td><td colspan="2">17,012 ms</td></tr>
<tr><td colspan="2" rowspan="2">5,498 ms</td><td colspan="2">Train</td><td colspan="2">Train</td></tr>
<tr><td colspan="2">868 ms</td><td colspan="2">781 ms</td></tr>
<tr><td>Pre-evaluation (total)</td><td colspan="2">26,384 ms</td><td colspan="2">177,387 ms</td><td colspan="2">38,766 ms</td></tr>
<tr><td rowspan="2">Evaluation</td><td>Cross-validation</td><td>Compute score</td><td>Cross-validation</td><td>Predict</td><td>Cross-validation</td><td>Predict</td></tr>
<tr><td>33,044 ms</td><td>9 ms</td><td>7,813 ms</td><td>< 1 ms</td><td>6,216 ms</td><td>< 1 ms</td></tr>
<tr><td>Total</td><td>59,428 ms</td><td>26,391 ms</td><td>185,200 ms</td><td>177,387 ms</td><td>44,982 ms</td><td>38,766 ms</td></tr>
</table>

Table 17. Execution time experimentation results (BM25F and FE-SVM)

Therefore, it can be concluded that from a performance perspective, the FE-SVM approach gives better results than the BM25F approach.

8. Conclusions

This final chapter provides a general overview of the project, its development and its results. This includes an evaluation from the thesis management development perspective and an analysis of the goals and the generated results.

8.1. Objectives achievement

This section enumerates and justifies the satisfaction and the achievement of each one of the specific goals depicted during the work plan, according to the work presented in this thesis final document.

[O1.1.] ***To study the current status of similarity detection*** *in the RE field from a general point of view.*

A systematic literature review has been depicted in chapters 2 and 3, including all the steps that have been performed and the data extraction and synthesis techniques that have been used as input information for the development of this thesis.

[O1.2.] ***To review and to enumerate similarity detection techniques/algorithms****, and to be specific the ML and NLP techniques that represent the state-of-the-art of the field.*

Chapter 3 is a general, summarized overview of the different approaches and techniques reviewed during the systematic literature review. This review is presented as a synthesized depiction of the most representative techniques and the algorithmic evaluation used to decide the algorithms to be developed.

[O1.3.] ***To identify potentially suitable algorithm candidates*** *for this master thesis and the use case with which it will be validated.*

Section 3.4, and to be specific Table 5, report the results of these candidate algorithms evaluation and comparative analysis to select those approaches to be developed as part of this master thesis. The selection criteria and the justifications are depicted in this section.

[O2.1.] ***To elaborate a development proposal*** *for the implementation of the selected algorithms.*

Chapters 5 and 6 are structured with two main report sections. The first one of each chapter is a theoretical, analytical depiction of the technical details of each algorithm. The second section of each chapter is a depiction of the development process, the adaptation of the algorithm to the defined use case, and the different steps and techniques used.

[O2.2.] ***To integrate the algorithms with a unique tool*** *to use and to test the different similarity detection scenarios.*

Section 4.1.1, and to be specific Figure 4, provide a general overview of the Requirements Similarity system. This includes the analysis of integrated features between both algorithms (i.e., the requirements data management), as well as the features deployed by each algorithm using this interface tool, as depicted in sections 5.3 and 6.3, where each algorithm features are developed.

[O3.1.] ***To evaluate the requirements of the input data of the algorithms****, in order to guarantee a comprehensive analysis of the results.*

The depiction of the Requirement Similarity system in Figure 4 includes how the developed system manages the export of requirements data. Additionally, and as part of the OpenReq project, the requirements data schema is depicted in section 4.1.2, in alignment with the available data in the Qt's issue repository, which is used to validate the algorithms.

[O3.2.] ***To optimize and adapt the algorithms*** *based on the use case requirements.*

As depicted before, sections 5.2 and 6.2 provide all development details regarding the adaptation of each algorithm, previously described, to the dataset and the validation use case.

[O3.3.] ***To analyze and to prepare a dataset of the use case*** *for all scenarios (i.e., all the different similarity detection algorithms).*

In chapter 7, where the Experimentation stage is developed, the details about the use case and the experimentation dataset details are presented.

[O3.4.] ***To carry out the experiments*** *using the developed algorithms.*

A set of experiments are designed and reported in section 7.5. Additionally, some additional results are included in Annex B, especially for those optimization processes requiring large amounts of data to be processed.

[O3.5.] ***To perform a comparative analysis*** *between algorithms.*

A comparative analysis is developed from two perspectives: accuracy or reliability, and performance or efficiency. A comparative schema is built by adapting the results of each algorithm so that they can be compared in section 7.5.

[O3.6.] ***To extract conclusions in terms of the reliability*** *of the results* ***and the performance*** *of the algorithms.*

The reliability comparison is presented in section 7.5.1, while the performance analysis is presented in section 7.5.2. They are developed separately to evaluate the benefits and disadvantages of each algorithm from different perspectives. A general conclusion is provided in section 8.2.

The results presented in this memory and the Requirements Similarity system justify the achievement of all planned objectives.

8.2. General project evaluation

After the achievement of the previous objectives, it can be justified that this master thesis development has generated three main results:

1. A **systematic literature review** which analyzes the state-of-the-art of the similarity detection field in RE. This review includes the most common NLP and ML techniques used for processing natural language information and other metadata fields. Furthermore, some of the most frequent general approaches for duplicate detection have been presented and analyzed, including vector-based approaches using TF-IDF variations (BM25F) and align-based approaches with feature extraction and supervised classification processes (FE-SVM approach).

2. The **development of a Requirements Similarity system** which exposes and provides a strong infrastructure for the analysis of different algorithm approaches and the development and usage of some of the most representatives techniques of the field.

3. A **comparative, qualitative analysis in accuracy and performance** of the two developed algorithms, including a real use case scenario and detailed data about the reliability of the results and the performance thanks to the availability of training and validation data.

The experimentation results depicted in chapter 7 prove that the selection of one algorithm or another will depend on the use case and the requirements of the software engineering scenario. In terms of reliability, the BM25F approach has proven to be the most accurate solution in detecting duplicate pairs of requirements for the given use case. However, if the analysis is focused on performance and efficiency, the FE-SVM provides better results.

To choose an algorithm for a specific use case is as important as its development. Software engineers must be able to identify the requirements of their scenarios and to select the algorithm that suits better.

8.3. Future work

After the development and the conclusions depicted in this master thesis, there are some ideas and future work lines that could be exploited to improve the Requirements Similarity system. These include:

- To **improve the DB access management** by providing better read/update/write/delete functionalities. Some minor design considerations, like the creation of a database index, have already been addressed. However, as RE requires the management of large amounts of data, it seems accurate to explore different approaches and techniques to improve

- To allow **concurrency between similarity evaluation techniques**. The system is not currently designed to fully handle concurrent requests for all features, especially when working with subsets of the requirements data (i.e., data belonging to specific projects). A significant improvement, especially to be applied and used in real scenarios as a component and not a research tool, would be to provide and develop the required features to successfully run all kinds of parallel executions.

- To **extend the current execution with new algorithms** and evaluation techniques. The design and the development of the system have been focused on providing an adaptive, extensive structure to reuse and extend new features and algorithms using different NLP and ML techniques. It would be interesting to maintain this component with up-to-date solutions and new approaches that could improve the current state-of-the-art in the similarity detection field.

These lines of work would not affect the results provided in this master thesis, but they would improve the performance and the usage of the Requirements Similarity system, which is the main software result delivered as part of the work that has been developed in this document.

A. Glossary of terms

AI	Artificial Intelligence
ML	Machine Learning
NLP	Natural Language Processing
SE	Software Engineering
RE	Requirements Engineering
IR	Information Retrieval
TF	Term Frequency
IDF	Inverse Document Frequency
SVM	Support Vector Machine
FE	Feature Extraction
PoC	Proof-of-Concept
RS	Requirement Similarity
POS	Part-of-Speech
SBD	Sentence Boundary Disambiguation
UC	Use Case
D	Duplicate
ND	Not-Duplicate

B. Bibliography

[1] Brynjolfsson, Erik, and Tom Mitchell. “What Can Machine Learning Do? Workforce Implications.” Science. American Association for the Advancement of Science. https://science.sciencemag.org/content/358/6370/1530.

[2] Collobert, Ronan, Jason Weston, Leon Bottou, Michael Karlen, Pavel Kuksa, and Koray Kavukcuoglu. “Natural Language Processing (Almost) from Scratch.” arXiv.org. https://arxiv.org/abs/1103.0398.

[3] Jeremy Dick, Elizabeth Hull, Ken Jackson, Requirements Engineering, Springer, pp. 7–9, ISBN 978-3-319-61073-3

[4] Natt och Dag, J., Regnell, B., Carlshamre, P. et al, A Feasibility Study of Automated Natural Language Requirements Analysis in Market-Driven Development - Requirements Eng (2002) 7: 20, https://doi.org/10.1007/s007660200002

[5] Tung Khuat, Nguyen Hung and Le Thi My Hanh. A Comparison of Algorithms used to measure the Similarity between two documents. International Journal of Advanced Research in Computer Engineering \& Technology (IJARCET). https://pdfs.semanticscholar.org/43f8/027780d2694331ca373c57f9a2ace509a7b6.pdf

[6] Yu Huang and Fei Chiang. Refining Duplicate Detection for Improved Data. http://ceur-ws.org/Vol-2038/paper3.pdf

[7] Requirements Engineering - Tools and Solutions Offered by OpenReq. OpenReq. https://openreq.eu/.

[8] Kniberg, Henrik, Mattias Skarin, and David Anderson. "Kanban y Scrum–obteniendo lo mejor de ambos." Prólogo de Mary Poppendieck \& David Anderson. ESTADOS UNIDOS DE AMÉRICA: C4Media Inc (2010).

[9] Schwaber, Ken, and Jeff Sutherland. “The Scrum Guide.” Scrum.org. https://www.scrum.org/resources/scrum-guide.

[10] Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2008). Systematic literature reviews in software engineering – A systematic literature review. Information and Software Technology, 51, 7–15. https://doi.org/10.1016/j.infsof.2008.09.009

[11] Elsevier. “What Is Scopus Preview?” What is Scopus Preview? - Scopus: Access and use Support Center. https://service.elsevier.com/app/answers/detail/a_id/15534/supporthub/scopus/#tips.

[12] “ACM Digital Library.” ACM Digital Library. https://dl.acm.org/.

[13] IEEE Xplore Help. https://ieeexplore.ieee.org/Xplorehelp/#/overview-of-ieee-xplore/about-ieee-xplore.

[14] “Explore Scientific, Technical, and Medical Research on ScienceDirect.” ScienceDirect.com | Science, health and medical journals, full text articles and books. https://www.sciencedirect.com/.

[15] Tarasov, D. S. (2015). Natural language generation, paraphrasing and summarization of user reviews with recurrent neural networks. Komp’juternaja Lingvistika i Intellektual’nye Tehnologii, 1(14), 595–602. Rossiiskii Gosudarstvennyi Gumanitarnyi Universitet.

[16] Jingjing Liu and Stephanie Seneff. 2009. Review sentiment scoring via a parse-and-paraphrase paradigm. In Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1 (EMNLP '09), Vol. 1. Association for Computational Linguistics, Stroudsburg, PA, USA, 161-169.

[17] Elsevier. “Reference Manager and Academic Social Network - Mendeley Database: Elsevier Solutions.” Reference Manager and Academic Social Network - Mendeley Database | Elsevier Solutions. https://www.elsevier.com/solutions/mendeley.

[18] Fu, C., An, B., Han, X., & Sun, L. (2016). ISCAS-NLP at SemEval-2016 task 1: Sentence similarity based on support vector regression using multiple features. SemEval 2016 - 10th International Workshop on Semantic Evaluation, Proceedings, 645–649.

[19] Xiaoyin, W., Lu, Z., Tao, X., Anvik, J., & Sun, J. (2008). An approach to detecting duplicate bug reports using natural language and execution information. Proceedings - International Conference on Software Engineering, 461–470. https://doi.org/10.1145/1368088.1368151

[20] Sun, C., Lo, D., Khoo, S. C., & Jiang, J. (2011). Towards more accurate retrieval of duplicate bug reports. 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE 2011, Proceedings, 253–262. https://doi.org/10.1109/ASE.2011.6100061

[21] El-Alfy, E.-S. M. (2014). Statistical analysis of ml-based paraphrase detectors with lexical similarity metrics. ICISA 2014 - 2014 5th International Conference on Information Science and Applications. https://doi.org/10.1109/ICISA.2014.6847467

[22] Mahajan, R. S., & Zaveri, M. A. (2017). Machine learning based paraphrase identification system using lexical syntactic features. 2016 IEEE International Conference on Computational Intelligence and Computing Research, ICCIC 2016. https://doi.org/10.1109/ICCIC.2016.7919721

[23] Magnolini, S., Feltracco, A., & Magnini, B. (2016). FBK-HLT-NLP at SemEval-2016 task 2: A multitask, deep learning approach for interpretable semantic textual similarity. SemEval 2016 - 10th International Workshop on Semantic Evaluation, Proceedings, 783–789.

[24] Kozareva, Z., & Montoyo, A. (2006). Paraphrase identification on the basis of supervised machine learning techniques. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).

[25] Yih, W. (2009). Learning Term-weighting Functions for Similarity Measures. Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2, 793–802. Retrieved from http://dl.acm.org/citation.cfm?id=1699571.1699616

[26] Kashyap, A., Han, L., Yus, R., Sleeman, J., Satyapanich, T., Gandhi, S., & Finin, T. (2016). Robust semantic text similarity using LSA, machine learning, and linguistic resources. Language Resources and Evaluation, 50(1), 125–161. https://doi.org/10.1007/s10579-015-9319-2

[27] Lee, M. C., Chang, J. W., & Hsieh, T. C. (2014). A grammar-based semantic similarity algorithm for natural language sentences. The Scientific World Journal, 2014. https://doi.org/10.1155/2014/437162

[28] Runeson, P., Alexandersson, M., & Nyholm, O. (2007). Detection of duplicate defect reports using natural language processing. Proceedings - International Conference on Software Engineering, 499–508. https://doi.org/10.1109/ICSE.2007.32

[29] Qiu, L., Kan, M.-Y., & Chua, T.-S. (2006). Paraphrase Recognition via Dissimilarity Significance Classification. Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing, 18–26. Retrieved from http://dl.acm.org/citation.cfm?id=1610075.1610079

[30] B. Dolan, C. Quirk, and C. Brockett, “Unsupervised Construction of Large Paraphrase Corpora: Exploiting Massively Parallel News Sources,” in Proceedings of the 20th International Conference on Computational Linguistics, 2004.

[31] System Dashboard - Qt Bug Tracker. https://bugreports.qt.io/secure/Dashboard.jspa.

[32] Atlassian. “Jira: Issue & Project Tracking Software.” Atlassian. https://www.atlassian.com/software/jira.

[33] “Swagger UI.” Swagger. https://swagger.io/tools/swagger-ui/.

[34] “Java SE Runtime Environment 8 Downloads.” Java SE Runtime Environment 8 - Downloads. https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html.

[35] “Spring Projects.” Spring. https://spring.io/projects/spring-boot.

[36] “SpringFox.” SpringFox by springfox. http://springfox.github.io/springfox/.

[37] MySQL. https://www.mysql.com/.

[38] Apache Lucene Core. https://lucene.apache.org/core/

[39] Extended Java WordNet Library. http://extjwnl.sourceforge.net/

[40] Team, The Apache OpenNLP. “Welcome to Apache OpenNLP.” Brand. https://opennlp.apache.org/.

[41] “The Stanford NLP Group.” The Stanford Natural Language Processing Group. https://nlp.stanford.edu/.

[42] Statistical Machine Intelligence & Learning Engine. https://github.com/haifengl/smile.

[43] Sun, C., Lo, D., Khoo, S. C., & Jiang, J. (2010). 2010 ACM/IEEE 32nd International Conference on Software Engineering. https://ieeexplore.ieee.org/document/6062072

[44] Company, The Qt. “The Qt Company.” The Qt Company. https://www.qt.io/company.

C. Annexes

A. JSON OpenReq Schema


```
{
  "requirement": {
    "$schema": "http://json-schema.org/draft-06/schema#",
    "title": "Requirement",
    "description": "A requirement within the OpenReq framework",
    "type": "object",
    "properties": {
      "id": {
        "description": "The unique identifier of a Requirement. Not a null value or an empty string.",
        "type": "string"
      },
      "name": {
        "description": "The name or title of a Requirement.",
        "type": "string"
      },
      "text": {
        "description": "The textual description or content of a Requirement. Note that Attachments (see below) can be used for other than plain text format descriptions.",
        "type": "string"
      },
      "requirementParts": {
        "description": "Aggregation of RequirementParts out of which the requirement consists of. This aggregation provides a mechanism for specifying requirement fragments or additional information for the Requirement.",
        "type": "array",
        "items": {
          "type": {
            "$ref": "#requirementPart"
          }
        }
      }
    }
  },
  "project": {
    "schema": "http://json-schema.org/draft-06/schema#",
    "title": "Project",
    "description": "A set of interrelated activities, carefully planned usually by a project team, to be executed over a fixed period of time and within certain cost and other limitations of resources, to implement a certain software system.",
    "type": "object",
    "properties": {
      "id": {
        "description": "The unique identifier of a Project. Not a null value or an empty string.",
        "type": "string"
      },
      "specifiedRequirements": {
```

```
      "description": "The set of Requirements that have been specified for the project.",
      "type": "array",
      "items": {
        "$ref": "#requirement"
      }
    }
   }
  },
  "dependency": {
   "$schema": "http://json-schema.org/draft-06/schema#",
   "title": "Dependency",
   "description": "A relationship between Requirements. All relationships are binary relations between two Requirements.",
   "type": "object",
   "properties": {
    "dependency_type": {
      "description": "The type of the Dependency.",
      "type": "string",
      "enum": [
        "duplicates"
      ]
    },
    "dependency_score": {
      "description": "An estimation of the reliability of the Dependency. The estimation is needed, e.g, when dependency is extracted by automation, such as natural language processing.",
      "type": "float"
    },
    "status": {
      "description": "The state of the Dependency in its life cycle.",
      "type": "string",
      "enum": [
        "proposed",
        "accepted",
        "rejected"
      ]
    },
    "fromid": {
      "description": "The requirement from which the Dependency originates.",
      "type": {
        "$ref": "#requirement"
      }
    },
    "toid": {
      "description": "The requirement to which the Dependency points to.",
      "type": {
        "$ref": "#requirement"
      }
    }
   }
  }
}
```

B. SVM configuration optimization results

kernel	C	sigma	TP	TN	FP	FN	Accuracy	Precision	Recall	F-measure
RBF	0,001	0,001	1236	488	987	159	60,07%	55,60%	88,60%	68,33%
		0,01	1390	70	1406	4	50,87%	49,71%	99,71%	66,35%
		0,1	818	572	904	576	48,43%	47,50%	58,68%	52,50%
		1	1254	147	1329	140	48,82%	48,55%	89,96%	63,06%
		10	779	737	738	616	52,82%	51,35%	55,84%	53,50%
		100	553	881	595	841	49,97%	48,17%	39,67%	43,51%
		1000	560	889	588	833	50,49%	48,78%	40,20%	44,08%
	0,01	0,001	1346	755	719	50	73,21%	65,18%	96,42%	77,78%
		0,01	1355	772	706	37	74,11%	65,74%	97,34%	78,48%
		0,1	1213	1282	195	180	86,93%	86,15%	87,08%	86,61%
		1	1049	1403	71	347	85,44%	93,66%	75,14%	83,39%
		10	800	755	722	593	54,18%	52,56%	57,43%	54,89%
		100	970	436	1039	425	48,99%	48,28%	69,53%	56,99%
		1000	821	573	901	575	48,57%	47,68%	58,81%	52,66%
	0,1	0,001	1371	633	841	25	69,83%	61,98%	98,21%	76,00%
		0,01	1283	1177	300	110	85,71%	81,05%	92,10%	86,22%
		0,1	1167	1317	161	225	86,55%	87,88%	83,84%	85,81%
		1	1101	1367	109	293	85,99%	90,99%	78,98%	84,56%
		10	772	1309	166	623	72,51%	82,30%	55,34%	66,18%
		100	542	869	606	853	49,16%	47,21%	38,85%	42,63%
		1000	696	735	739	700	49,86%	48,50%	49,86%	49,17%
	1	0,001	1336	875	600	59	77,04%	69,01%	95,77%	80,22%
		0,01	1220	1313	162	175	88,26%	88,28%	87,46%	87,86%
		0,1	1190	1336	142	202	88,01%	89,34%	85,49%	87,37%
		1	1131	1364	111	264	86,93%	91,06%	81,08%	85,78%
		10	1029	1411	64	366	85,02%	94,14%	73,76%	82,72%
		100	827	582	895	566	49,09%	48,03%	59,37%	53,10%
		1000	1096	441	1035	298	53,55%	51,43%	78,62%	62,18%
	10	0,001	1142	1334	142	252	86,27%	88,94%	81,92%	85,29%
		0,01	1189	1304	172	205	86,86%	87,36%	85,29%	86,32%
		0,1	1175	1338	137	220	87,56%	89,56%	84,23%	86,81%
		1	1115	1379	96	280	86,90%	92,07%	79,93%	85,57%
		10	1103	1131	344	292	77,84%	76,23%	79,07%	77,62%
		100	835	1159	315	561	69,48%	72,61%	59,81%	65,59%
		1000	0	1476	0	1394	51,43%	0,00%	0,00%	0,00%
	100	0,001	1329	924	552	65	78,50%	70,65%	95,34%	81,16%
		0,01	1224	1197	279	170	84,36%	81,44%	87,80%	84,50%
		0,1	1134	1233	244	259	82,47%	82,29%	81,41%	81,85%
		1	1115	1286	190	279	83,66%	85,44%	79,99%	82,62%
		10	1077	1239	236	318	80,70%	82,03%	77,20%	79,54%
		100	934	1421	53	462	82,06%	94,63%	66,91%	78,39%
		1000	974	439	1035	422	49,23%	48,48%	69,77%	57,21%
	1000	0,001	1331	915	560	64	78,26%	70,39%	95,41%	81,01%
		0,01	1261	965	509	135	77,56%	71,24%	90,33%	79,66%
		0,1	1102	1160	314	294	78,82%	77,82%	78,94%	78,38%
		1	1013	994	482	381	69,93%	67,76%	72,67%	70,13%
		10	1121	1338	139	272	85,68%	88,97%	80,47%	84,51%
		100	1047	1410	65	348	85,61%	94,15%	75,05%	83,53%
		1000	820	1420	54	576	78,05%	93,82%	58,74%	72,25%

LINEAR	0,001	-	683	881	595	711	54,49%	53,44%	49,00%	51,12%
	0,01	-	1084	1262	214	310	81,74%	83,51%	77,76%	80,53%
	0,1	-	1076	1388	88	318	85,85%	92,44%	77,19%	84,13%
	1	-	992	1363	112	403	82,06%	89,86%	71,11%	79,39%
	10	-	1115	1225	251	279	81,53%	81,63%	79,99%	80,80%
	100	-	1056	1274	200	340	81,18%	84,08%	75,64%	79,64%
	1000	-	865	573	903	529	50,10%	48,93%	62,05%	54,71%