RESim - Automated Detection of Duplicated Requirements in Software Engineering Projects

Quim Motger
ESSI Dept.
jmotger@essi.upc.edu

Cristina Palomares
ESSI Dept.
cpalomares@essi.upc.edu

Jordi Marco
CS Dept.
jmarco@cs.upc.edu

Software and Service Engineering Group
Universitat Politecnica de Catalunya - BarcelonaTech

Abstract

Collaborative software development experience in recent years proves that the management of large sets of requirements has become a critical issue. Among the main problems of requirements engineering, the detection and management of duplicated requirements is highlighted. If ignored, these redundancies may lead to the duplicity of tasks, which is a hazardous issue from a project management perspective. Moreover, the automation of this process and the standardized use of accurate, open-source tools are still at a state-of-the-art stage. Based on this scenario, we introduce RESim - a software development proposal which integrates different techniques for the detection of duplicated requirements in an adaptive, scalable tool. RESim solution is based on a twofold objective approach: first, to deliver an easy-to-use, practical tool for duplicated requirements detection for requirements engineers; second, to provide an evaluation framework of different similarity detection algorithms for researchers. A video demonstration of a GUI component developed for user testing purposes is available here: https://youtu.be/A7dnLgWInMs.

1 Introduction

One of the key challenges of the requirements engineering field is the adaptation to new software development scenarios alongside the exploitation of the features of these new environments. The state-of-the-art is evolving and applying last innovation techniques from Natural Language (NL) and Machine Learning (ML) fields to the autonomous management and evaluation of requirements textual data [1]. On the other hand, Open-Source Software (OSS) development environments are responsible of an evolution in traditional tasks such as the report, the identification and the analysis of new features and new requirements in software engineering projects [5]. In this new context, this information is typically reported by multiple sources among the variety of stakeholders, including developers, managers and end-users. 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.

Among these evaluation tasks, the detection of duplicated requirements in software engineering projects has been object of research during the last decade. Based on the definition by Natt et. al. [7], a pair of **duplicated requirements** can be conceived as two requirements with a level of similarity to the extent that one of them can be eliminated without losing relevant information. Redundant information in the requirements set of a project can lead to the duplicity of tasks, which is a significantly costly issue from a project management perspective.

In: M. Sabetzadeh, A. Vogelsang, S. Abualhaija, M. Borg, F. Dalpiaz, M. Daneva, N. Fernández, X. Franch, D. Fucci, V. Gervasi, E. Groen, R. Guizzardi, A. Herrmann, J. Horkoff, L. Mich, A. Perini, A. Susi (eds.): Joint Proceedings of REFSQ-2020 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track, Pisa, Italy, 24-03-2020, published at http://ceur-ws.org

Despite this, it is difficult to find open source tools providing generic, adaptive solutions for the automated detection of duplicated requirements and most of them are addressed to a very specific use case [3].

Based on a systematic literature review of the textual similarity detection field, we design and develop the RESim tool - a software development proposal served from NL and ML up-to-date techniques for the automated detection of duplicated requirements in large software engineering projects. A twofold objective baseline approach has been followed for the design and the development of the RESim tool.

Objective 1. To develop an easy-to-use, independent tool for the autonomous detection of duplicated requirements that allows its integration in multi-platform environments for requirement engineers as end-users.

Objective 2. To deliver a software system proposal for the evaluation of multiple textual similarity detection algorithms. For this purpose, it is necessary to design a loosely coupled architecture which allows researchers to easily integrate new duplicated detection functionalities.

RESim development and its evaluation have been carried out within the scope of the Horizon 2020 OpenReq project[1], whose goal is to deliver intelligent recommendation and decision technologies for the requirements engineering community.

2 Background

This section is a summary report of the related work studied by the systematic literature review results detailed by Motger's MSc thesis [6], one of the co-authors of this paper.

2.1 Classification of similarity detection approaches

Based on the NL similarity detection state-of-the-art techniques, we define 3 general algorithmic approaches for similarity evaluation.

Align-based approach. NL information is used to compute a similarity score based on the alignment (i.e., the intersection or resemblance between these features) of this NL representation information.

Vector-based approach. NL textual items are converted into a vector representation, i.e., a bag of words or tokens with additional information like the frequency or the position of a specific token. Using this new representation, a similarity score is obtained using vector-based metrics which evaluate the grade of intersection between these vectors.

Feature extraction with supervised classification. A set of NL features from each pair of requirements is extracted and used for a supervised classification process to predict whether the requirements are duplicates (D) or not (ND).

2.2 Selection of similarity detection algorithms

Among the state-of-the-art similarity detection algorithms, two different approaches are developed within the RESim system: a vector-based representation using the BM25F model, and an align-based feature extraction process with supervised classification. In this section we develop the theoretical details of these techniques.

Vector-based representation: an extension of BM25F model. The first similarity algorithm (BM25F) is based on an extension of the BM25F model as described by Sun et. al. [8]. BM25F is an Information Retrieval (IR) model used to find relevant documents inside a large data set of documents or *corpus* given a relatively short text query. This model is based on two numerical statistics: the *Inverse Document Frequency* (IDF), which computes the global importance of a term t among the *corpus* of documents based on the inverse frequency value; and the *local Term Frequency* (TF), which computes the local importance measure of a term t in a specific document D based on the direct frequency value. BM25F proposes a combination of both IDF and TF for the intersection tokens using a bag-of-words representation of the NL fields. This combination provides a similarity score which can be use to provide a ranked list of the most similar documents of a specific document.

Align-based feature extraction with supervised classification. The second similarity algorithm (FESVM) is based on a Feature Extraction (FE) process using align-based features and a Support Vector Machine (SVM) classifier, based on the work presented by Mahajan et. al. [4]. During the feature extraction stage, the algorithm processes a set of document pairs. For each pair, a set of NL features is extracted - typically, numerical features based on the level of resemblance between the two processed documents. Among these features we identify lexical features (i.e., word-to-word match, bi-gram match) and syntactic features (i.e., subject-to-subject match, verb-to-verb match). There is no consensus over the state-of-the-art regarding the usage of syntactic features in duplicated detection scenarios. Mahajan et. al. claim to improve results by combining these features. However, El-Alfy et. al. [2] focus exclusively in lexical features.

[1] OpenReq - https://openreq.eu/

3 RESim approach

3.1 General overview

The RESim tool is an adaptive system for multi-algorithmic integration of different similarity detection techniques, for the final purpose of providing a usable tool for the identification of duplicated requirements. Based on the two objectives depicted in section 1, the system must satisfy a set of predefined requirements.

R1. To manage (store and read) software project requirements data internally.

R2. To integrate different similarity detection algorithms in a scalable environment, defining an adaptive architecture motivated by the possibility of extending the system with new similarity detection algorithms.

R3. To provide a generic, unique interface via a REST API deployment of the tool with a unique data schema for the import and export of requirements data.

R4. To expose empirical evaluation features for the different developed algorithms.

R5. To provide access to reliable and efficient duplicate detection algorithms which can be applied in real software engineering scenarios.

3.2 Software architecture

To provide an adaptive, scalable environment, we define a software architecture based on a horizontal and a vertical dimension. This is depicted in Figure 1.

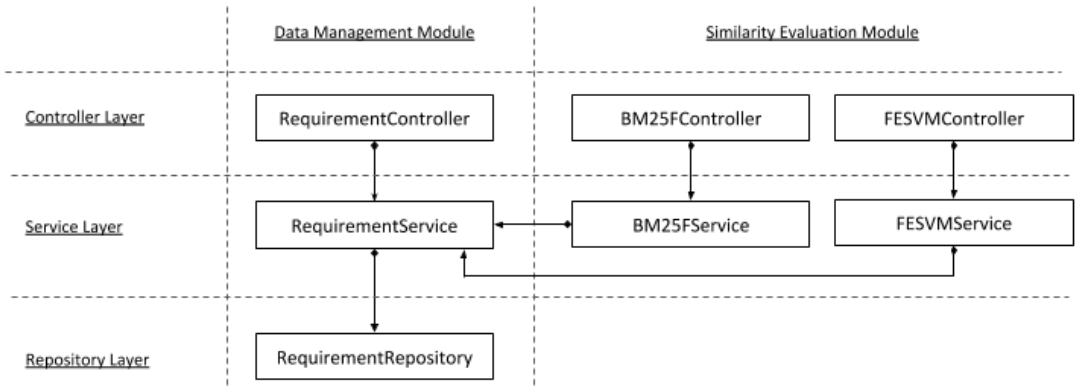

Figure 1: RESim software architecture

The vertical dimension depicts a 3-layer architecture: the *Controller Layer*, which implements and exposes the REST API of the service to fully deploy all similarity features providing a generic, easy-to-use interface; the *Service Layer*, which implements the core business-logic of the system; and the *Repository Layer*, which handles the data management of the requirements instances.

The horizontal dimension differentiates two decoupled modules. The first one is the *Data Management* module, which vertically integrates all components (controllers, logic and data access) related to the requirements data management. This module can be extended or used as a black box by the *Similarity Evaluation Module*, which develops the service logic related to the similarity algorithms. The purpose of this architecture is to provide a generic framework for the future extension of new similarity detection algorithms.

3.3 Business logic: autonomous detection of duplicated requirements

In this section we develop the similarity evaluation algorithms developed at RESim based on the selection of similarity detection techniques from Section 2.2.

(1) BM25F algorithm. The first algorithmic approach is based on an extension of the BM25F model by applying the local term frequency measure to the textual fields of a specific pair of requirements (i.e. duplicate candidates). We propose and develop a set of contributions to the algorithm depicted by Sun et. al.

First of all, the design and development of a specific **lexical NL pipeline** for the pre-processing of the NL fields of the requirements. This pipeline concatenates a series of lexical NL techniques which are applied to every requirement imported to the RESim system for performance purposes. Two textual fields are used for each requirement: the *name* or summary and the *text* or description. To each field, we apply a *sentence boundary disambiguation* (SBD) process, a *stop-word removal*, a *tokenizer*, a *lower-case filter* and a *stemming* process. As a result, the basic NL pipeline returns two bag-of-words representations of the NL fields.

Second of all, the selection of a specific **feature data set** (including NL fields and additional metadata fields), based on generic requirements metadata fields and the available data schema for empirical evaluation. The overall similarity function computed for each requirement pair (R_1,R_2) is defined by Equation 1.

- f_1 **Uni-gram score**. The BM25F score value using requirement name and text uni-grams.
- f_2 **Bi-gram score**. The BM25F score value using requirement name and text bi-grams.
- f_3 **Project score**. Set to 1 if R_1 and R_2 belong to the same project; 0 otherwise.
- f_4 **Type score**. Set to 1 if R_1 and R_2 are of the same type (bug, feature,...); 0 otherwise
- f_5 **Component score**. Computed by the Jaccard similarity between R_1-R_2 sets of components
- f_6 **Priority score**. Reciprocal distance between the priority of R_1-R_2 mapped to numerical values.
- f_7 **Versions score**. Reciprocal distance between R_1-R_2 latest version mapped to numerical values.

$$sim(R_1, R_2) = \sum_{i=1}^{7} w_i * f_i \tag{1}$$

Finally, the development of a **gradient descent optimization process** based on the weight (w_i) of each feature (f_i). This process is run using a data set of requirement pairs labelled as duplicates. For each duplicate pair, a third non-related requirement is randomly selected, building a triplet of requirements. The optimization process is then based on maximizing the difference between the partial derivatives of the duplicate related *sim* score and the not-duplicated *sim* score, applying at each iteration a small variation of a specific parameter.

Based on a *cross-validation* analysis using duplicate and not-duplicate labelled data, it is possible to extract a *threshold value* to discern between duplicated pairs (i.e., similarity values above the threshold) and not-duplicated pairs (i.e., similarity values below the threshold).

(2) FESVM algorithm. The FE-SVM approach is built upon a feature extraction process using align-based features between requirement pairs and a supervised classification process for the featured entities, as depicted in Section 2. Similarly to the BM25F algorithm, we propose a specific development approach to integrate the algorithm into the RESim tool.

First of all, a custom **syntactic NL pipeline** built for the syntactic feature extraction process. The basic NL pipeline is used in both algorithmic approaches. The bag-of-words representations of the *name* and *text* fields are used to compute the lexical alignment features between the pair of requirements. For the syntactic alignment features, it is necessary to apply an additional NL pipeline which applies a dependency parser to build the dependency tree structure, which include grammar structures and relations of each sentence. This syntactic NL pipeline applies a *POS tagger* and a *lemmatizer* to the bag-of-words of each requirement field. To this formatted bag-of-words, the pipeline builds a dependency tree using a *dependency parser* for each sentence.

Second of all, the selection of **lexical and syntactic features** based on the state-of-the-art review and the available data schema. Using the bag-of-words and the dependency tree representations, we compute the align-based feature values. Each feature is computed separately for the *name* and the *text* field - therefore, each feature f is reported as f_i for the *name* field and as f_{i+1} for the *text* field. Given a pair of requirements (R_1,R_2), features 1-6 refer to lexical features; features 7-14 refer to syntactic features.

- $f_{1\text{-}2}$ **Word overlap**. Computes the ratio of overlapping words or tokens between the R_1 and R_2 original texts. The score is computed using the Jaccard similarity function.
- $f_{3\text{-}4}$ **Uni-gram match**. Computes the Jaccard similarity between the bag-of-words sets of R_1 and R_2.
- $f_{5\text{-}6}$ **Bi-gram match**. Analogue to $f_{3\text{-}4}$, but using bi-grams as the elements to match between R_1 and R_2 bag-of-words sets. Similar to the bi-gram extraction process in the BM25F approach, sentence boundary information is used to avoid creating false bi-grams belonging to different sentences.
- $f_{7\text{-}8}$ **Subject match**. For each requirement in the pair, the subject/s of each sentence are extracted, i.e., the algorithm looks for the node/s 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.
- $f_{9\text{-}10}$ **Subject-verb match**. The algorithm looks in each sentence for all grammar patterns matching a dependency relation between a subject and a verb and its dependency label. 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 the requirements share a subject, a verb and the dependency label value that connects them.

$f_{11\text{-}12}$ **Object-verb match**. Similar to $f_{9\text{-}10}$, but the governor of the relationship must be an object instead of a subject. Jaccard similarity score and matching criteria are computed analogously.

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

Finally, the selection and optimization of an SVM classifier, which Mahajan et. al. report to be the best option for these scenarios. The set of featured pair entities are used to either train and test an SVM classifier using a Gaussian kernel configured with the most optimal parameters for the empirical evaluation.

4 Empirical Evaluation: Qt's use case results

For the validation of RESim, we have used the data of the Qt's public JIRA issue repository[2]. The Qt company is one of the OpenReq project partners. Table 1 provides a requirement example instance of the Qt's data set.

Name	Text	Project	Component	Type	Version	Priority
Debugger shows wrong address for pointer treated as array.	Take a pointer and try to change the Local Display Format to show it as an Array of 10 items.[...]	QTCREATORBUG	Debugger	Bug	4.2.1	NE (Not Evaluated)

Table 1: Requirement data example from Qt's issue repository data set

For each algorithm, we run a 10×10 cross-fold validation analysis using the 2,935 instances of labelled data. Notice that for the BM25F it is necessary to use the whole requirement *corpus* of 111,143 requirements to compute frequency values and to apply performance analysis in a real scenario.

Table 2 summarizes Qt's issue repository data used for validation. Table 3 summarizes the reliability experimentation results of both algorithms. Table 4 reports the execution time required by each general stage of both algorithms: the NL Pipeline (NLP); the Algorithm Data Structure generation (ADS); and the Computation Score or Prediction (CS/P). Notice that for the FE-SVM approach, only lexical features results are provided, as syntactic features did not report better results than using only lexical features.

Requirements	111,143
Projects	20
Labelled D	1,436
Labelled ND	1,499
Labelled total	2,935

Table 2: Evaluation data

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

Table 3: Reliability results

	BM25F	FE-SVM
NLP	20,886ms	20,886ms
ADS	5,498ms	17,793ms
CS/P	9ms	<1ms

Table 4: Performance results

As reported in Table 3, although the BM25F approach proves to be more accurate in terms of reliability in all metrics, both algorithms report similarly reliable values. The *f-measure* reported by the FE-SVM approach is 8% higher than the results reported by the Mahajan et. al. empirical evaluation. This comparison is restricted due to the differences between both experimentation scenarios, i.e. the data set used for evaluation. For BM25F, it is not possible to provide a numerical comparison, as their approach is not focused on a classification solution but on a similarity ranked list retrieval, and therefore they do not provide classification metrics.

If we focus on performance, the FE-SVM approach proves to be very efficient at prediction time (<1 ms). In large projects, prediction time is critical, as each new requirement needs to be tested against thousands of already existing requirements and, hence, the difference in CS/P execution time is significant. As a conclusion, in terms of performance, the FE-SVM approach proves to be more efficient than the BM25F approach.

5 Demo plan

This section depicts a plan for a demonstration of the tool. First, the RESim GUI component is introduced - a Java Swing based graphic component designed to test some of the main features of the RESim tool. Second, we depict the workflow for a demonstration scenario.

5.1 RESim GUI component

To facilitate and to demonstrate the functionalities of the RESim tool, a basic Swing Java-based application has been developed to expose the main features of the RESim system, including the management of requirements data and the evaluation of duplicated detection algorithms. The GUI component is deployed as an isolated interface tool which serves from the RESim REST API exposed features.

[2]Available at https://bugreports.qt.io/

5.2 Tool demo workflow

To use and evaluate RESim, this section depicts the user interaction to repeat the demo shown in the video attached in this publication (see Abstract). As a reference, please see the Github repositories of the RESim tool and the RESim GUI component[3] for software and developers support documentation.

1. Build the requirements data set. Build a JSON OpenReq schema instance with the list of requirements to be imported. Use the Swagger REST documentation as a reference for the OpenReq data schema.

2. Deploy and run the tool. Build and run the RESim service and the RESim GUI tools following the "*How to build*" and "*How to run*" sections in the README files available at the Github repositories.

3. Load the requirements. Use the RESim GUI component to select the JSON file containing the requirement list and to import them to the RESim tool using the 'Upload' button.

4. Select a project. Select a specific project from the dropdown list to show the list of related requirements.

5. Select and configure an algorithm. Mark the checkbox of the BM25F or the FE-SVM algorithms and configure the additional parameters. For the BM25F, it is necessary to define a duplicate threshold score. For the FE-SVM, it is necessary to indicate whether to use lexical features, syntactic features or both.

6. Load training data. Use the 'Upload' button to upload the list of labelled pairs used for training.

7. Train/optimize the tool. Click on the 'Train' button to optimize (BM25F) or train (FE-SVM) the algorithm using the previously imported training data set.

8. Load testing data. Use the 'Upload' button to upload the list of requirement pairs used for testing.

9. Test/predict duplicated requirements. Click on the 'Test' button to compute the score (BM25F) or predict the D/ND relation (FE-SVM) of each one of the requirement pairs.

6 Conclusions and future work

Empirical evaluation proves that RESim tool provides accurate and efficient state-of-the-art solutions to the problem of requirements duplicated detection. It paves logic to continuously evolve these algorithmic development, either by extending the system with new, up-to-date similarity evaluation solutions or by evaluating the tool performance in a wide variety of data sets and scenarios. Additionally, it would be interesting to study how end-user feedback can be used by the tool to learn about the validation of predicted duplicate/not-duplicate pairs, in order to use this knowledge to improve the algorithms reliability.

Acknowledgments

The work presented in this paper has been supported by the GENESIS project under the National Spanish Program for Research Aimed at the Challenges of Society (RETOS) 2016, contract TIN2016-79269-R.

References

[1] F. Dalpiaz, A. Ferrari, X. Franch, and C. Palomares. Natural language processing for requirements engineering: The best is yet to come. *IEEE Software*, 35(5):115–119, 2018.

[2] E.-S. M. El-Alfy. Statistical analysis of ml-based paraphrase detectors with lexical similarity metrics. *2014 International Conference on Information Science & Applications (ICISA)*, 2014.

[3] T. Khuat, N. Hung, and L. 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)*, 4:1117–1121, 04 2015.

[4] R. S. Mahajan and M. A. Zaveri. Machine learning based paraphrase identification system using lexical syntactic features. *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 2016.

[5] I. Mistrik, J. Grundy, A. V. D. Hoek, and J. Whitehead. Collaborative software engineering: Challenges and prospects. *Collaborative Software Engineering*, page 389–403, 2010.

[6] Q. Motger. Automated similarity detection: Identifying duplicated requirements. http://openaccess.uoc.edu/webapps/o2/handle/10609/105807, 2019.

[7] J. N. och Dag, B. Regnell, P. Carlshamre, M. Andersson, and J. Karlsson. A feasibility study of automated natural language requirements analysis in market-driven development. *Requirements Engineering*, 7:20–33, 2002.

[8] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, 2011.

[3] Available at https://github.com/quim-motger/requirements-similarity