Requirements Classification for Traceability Link Recovery

Tobias Hey
Karlsruhe Institute of Technology
Karlsruhe, Germany
hey@kit.edu

Jan Keim
Karlsruhe Institute of Technology
Karlsruhe, Germany
jan.keim@kit.edu

Sophie Corallo
Karlsruhe Institute of Technology
Karlsruhe, Germany
sophie.corallo@kit.edu

***Abstract*—Being aware of and understanding the relations between the requirements of a software system to its other artifacts is crucial for their successful development, maintenance and evolution. There are approaches to automatically recover this traceability information, but they fail to identify the actual relevant parts of the requirements. Recent large language model-based requirements classification approaches have shown to be able to identify aspects and concerns of requirements with promising accuracy. Therefore, we investigate the potential of those classification approaches for identifying irrelevant requirement parts for traceability link recovery between requirements and code.**

We train the large language model-based requirements classification approach NoRBERT on a new dataset of requirements and their entailed aspects and concerns. We use the results of the classification to filter irrelevant parts of the requirements before recovering trace links with the fine-grained word embedding-based FTLR approach.

Two empirical studies show promising results regarding the quality of classification and the impact on traceability link recovery. NoRBERT can identify functional and user-related aspects in the requirements with an F_1-score of 84%. With the classification and requirements filtering, the performance of FTLR could be improved significantly and FTLR performs better than state-of-the-art unsupervised traceability link recovery approaches.

***Index Terms*—Requirements Classification, Traceability Link Recovery, Requirements Engineering, Machine Learning, Information Retrieval, Language Model**

I. Introduction

Understanding the relations between the artifacts of software systems is crucial for their efficient development, maintenance, and management. The traceability of these relationships enables, for instance, the comprehension of design decisions or the assessment of the consequences of modifications [1]. However, generating or preserving traceability information manually requires a high level of manual effort, entailing potentially high costs as human expertise is required to understand the relationships. This is the reason why this information is not readily accessible in the majority of software projects. However, if we could make traceability information readily accessible by generating it automatically, the development, maintenance, and management of a wide range of software systems could become more efficient [1], [2]. Moreover, in safety-critical systems, such as aerospace or automotive, the elicitation of traceability information is explicitly mandated by regulatory bodies [3]–[5]. Existing approaches to automatically recover those trace links (TLs) between requirements and source code are not able to bridge the semantic gap between the artifacts. They achieve too low precision at acceptable recall levels to be used in practice [6]–[8].

Use case name: AdvancedSearch

Description: The tourist searching for a site using the potential offered by the Advanced Search

Participating actor: initialized by Tourist

Entry conditions: The Tourist has successfully authenticated to the system.

Flow of events User System:

1) Enable the advanced search feature from your personal area.
2) View the advanced search form.
3) Fill in the form of advanced search and submit.
4) Gets the position of relying on the tourist event of the use location and process the request.

Exit conditions: The system displays a list of results. Interruption of the connection to the server ETOUR.

Quality requirements: The system requirements into the transaction in more than 15 seconds.

Fig. 1. Example of an use case description from the eTour dataset that follows a template structure. Orange parts can be filtered by template labels. The red part can only be identified by an automatic classification.

Requirement 17:
An EST server MAY provide service for multiple CAs as indicated by an OPTIONAL additional path segment between the registered application name and the operation path.

The EST server MUST provide services regardless of whether the additional path segment is present.

Fig. 2. Example of a natural language requirement from the LibEST dataset. The red part describes a quality aspect of the system.

An important factor determining precision is selecting relevant parts of the requirements for establishing TLs. Approaches have shown improved performance when filtering certain parts of the requirements based on use case template labels [9]. However, even template elements that are regarded as relevant for the mapping to source code can contain

irrelevant information. For example in Figure 1, the flow of events description of the use case includes the sentence "Fill in the form of advanced search and submit," which describes a user interaction within the use case. This interaction cannot be directly mapped to the system's functionality, such as a method or class, as the act of filling out a form is not a functionality of the system. Similarly, requirements not following any template can also contain irrelevant parts. For example, the requirement in Figure 2 includes the statement "The EST server MUST provide services regardless of whether the additional path segment is present," which states a quality aspect but does not describe a specific system functionality. Therefore, selecting mappable and relevant parts of requirements can be important. If these parts are not filtered, they can lead to erroneously identified connections between artifacts and, thereby, reduce the precision of the traceability link recovery (TLR).

Determining the content, type, and category of requirements is a common research topic in requirements engineering. Recently, requirements classification approaches utilize machine learning techniques for classifying requirements. This includes tasks such as identifying requirements in requirement documents [10], [11], categorizing entire requirements into functional or non-functional [12]–[14] or subcategories thereof, as well as determining the present concerns or aspects in parts of the requirements [15], [16]. The information generated by such processes, thus, holds the potential to gain a more precise understanding of the requirements. This understanding, in turn, could be utilized to filter parts of the requirements and, thereby, improve the precision of automated TLR approaches.

Assessing this potential is the goal of this work. For this purpose, we manually labeled the parts of requirements in datasets commonly used in requirements-to-code TLR to provide a gold standard. Then we measure the performance of one of the state-of-the-art requirements classification approaches, NoRBERT [16], on these datasets. We use the classification results of NoRBERT to filter parts of the requirements and assess the impact on the performance of the requirements-to-code TLR approach FTLR [9]. Our contributions are:

1) A novel dataset for classifying (parts of) requirements
2) An empirical study on the performance of requirements classification approaches on TLR datasets
3) An empirical study on the effect of filtering requirements for fine-grained requirements-to-code TLR approaches

We publish the source code and further results in our supplementary material [17].

II. Foundations

This work builds upon the requirements classification approach NoRBERT and the requirements-to-code TLR approach FTLR that we further describe in this section.

A. NoRBERT

The classification model NoRBERT [16], [18] is based on the large language model BERT [19] that is pre-trained on the English Wikipedia and the BooksCorpus [20]. NoRBERT can utilize either of the available pre-trained BERT models:

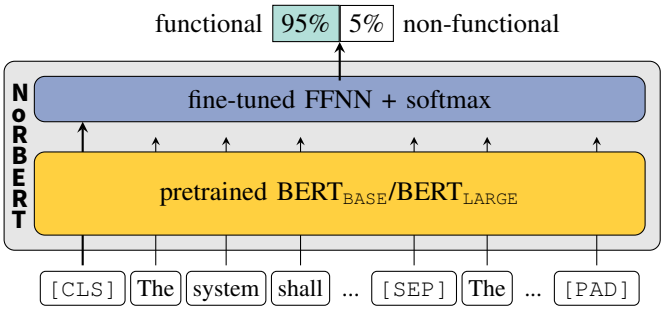

Fig. 3. Visualization of applying NoRBERT to requirements classification

the BERT$_{BASE}$ or the BERT$_{LARGE}$ model. Additionally, pre-trained tokenizers are available for these models, which are the only preprocessing steps utilized by NoRBERT. To use one of these models for the actual task of requirements classification, an additional layer is added. This layer takes the aggregated output of the last BERT layer (the output of the `[CLS]` token) as input (cf. Figure 3). It consists of a single-layer feedforward neural network (FFNN) that calculates the output from the sum of weighted inputs (and a certain bias). NoRBERT employs the softmax function to determine the probability distribution of the different classes.

Fine-tuning the model is accomplished by training the entire model on the specific classification task. The authors applied NoRBERT to the PROMISE NFR (PNFR) dataset [21]. Regarding subclassification of non-functional requirements and detection of functional and quality aspects, NoRBERT significantly improved F_1-score performance compared to previous methods. NoRBERT also outperformed existing methods when applied to projects that are not seen during training. Additionally, the authors examined the requirements of the PNFR dataset concerning the three aspects of functional requirements: function, data, and behavior. They provided the first dataset and results for automatic subclassification of functional requirements, where NoRBERT achieved an average F_1-score of 77% in a setup relevant for transferability to unseen projects. On the PNFR dataset [21], the authors report an optimal number of epochs of 10 or 16 for binary classification and 32 or 50 for multi-class classification.

B. FTLR

FTLR [9] utilizes information retrieval (IR)-based similarity metrics on word embeddings to retrieve TLs between requirements and source code classes. However, FTLR does not compare the whole artifacts but uses a fine-grained approach. In FTLR, both types of artifacts are split into smaller parts. Requirements are segmented into sentences. Use case descriptions following a template structure are split into their elements. Source code classes are split into their public methods which are represented by their signatures, extended by the name of the containing class and their associated comments. The elements of both artifact types undergo preprocessing steps such as stop word removal, lemmatization, and word length filtering. The resulting artifact elements are represented

by utilizing pre-trained word embedding representations from fastText [22].

The actual TLs are generated as follows. First, the similarity of the fine-grained elements is calculated based on the Word Mover's Distance (WMD) [23]. Then, the pairs undergo filtering using a fixed threshold to ensure a certain similarity between the elements. FTLR establishes TLs for a source code class by majority vote, associating it with the most frequently linked requirements among its methods. In this process FTLR can be configured to utilize call dependencies (`CD`), method comments (`MC`), and filter a fixed set of use case template elements (`UCT`). In their evaluation, the authors report the best results when using all three of these variants.

As FTLR already operates on fine-grained requirements elements, can filter requirements based on template structures, and has an accessible replication package, it is well suited to investigate the potential of requirements classification on requirements-to-code TLR.

III. Related Work

The publications related to our work can be classified into two categories: Approaches to automated requirements classification and requirements-to-code TLR approaches.

A. Requirements Classification

The automated extraction and classification of requirements from natural language text documents has been a focus of research for more than a decade. The approaches by Cleland-Huang et al. [12], [24] employ IR to identify non-functional requirements. They search for indicator terms in the documents. The approach is imprecise but can achieve high recall. The authors also published their dataset NFR [21] as part of the PROMISE Repository [25], which still serves as the standard benchmark dataset for requirements classification.

Other approaches utilize classical machine learning (ML) techniques such as decision trees [26], K-nearest neighbors [10], LDA [27], SVM [13], or naive Bayes [14] to classify the requirements of the PNFR dataset. They achieve F_1-scores of up to 95% on identifying non-functional requirements. Another line of research employs deep learning techniques to solve the problem. They utilize techniques such as convolutional neural networks (CNNs) [11], [28], [29], word embeddings [29], or document embeddings [30].

Dalpiaz et al. [15] relabel the PNFR dataset to address requirements encompassing both functional and quality aspects. They reimplement the SVM-based approach by Kurtanović and Maalej [13] and evaluate it on the relabeled dataset and other projects. Additionally, they propose a more interpretable set of features that yields lower but still comparable results.

Though promising, the aforementioned methods might not be practical for real-world use. They are often tailored to specific datasets, reliant on specific wording and sentence structure, or require manual preprocessing. These approaches either do not report their ability to generalize from project specifics or do not generalize well enough to apply to unseen projects as required in TLR. One reason for this might be the lack of available training data in requirements engineering to train more generalizable models. Furthermore, these works primarily focus on classifying non-functional requirements and distinguishing them from functional ones. A more detailed exploration of functional requirement aspects has been scarcely investigated.

These reasons led to the exploration of approaches that exploit the transfer learning capabilities of large language models (LLMs) for the task of requirements classification. This approach promises acceptable results and higher transferability to unseen projects, even for tasks with limited data, such as classifying functional aspects. Building on the promising results of NoRBERT (cf. Section II-A), subsequent works have employed BERT and transfer learning for requirements classification. For instance, Li et al. [31] combine BERT with a graph attention network (GAT) on dependency graphs of requirements. Their DBGAT model outperforms NoRBERT in classifying requirements into non-functional and functional and the four most common non-functional subclasses (usability, security, deployment, and performance). However, regarding the classification of all ten non-functional requirements subclasses in the PNFR dataset, the results of NoRBERT and DBGAT are similar: NoRBERT even surpasses DBGAT in classes with low occurrence in the dataset. Unfortunately, the authors do not provide insights into DBGAT's performance concerning functional and quality aspects or the sub-concerns of functional requirements.

Luo et al. [32] use prompt learning to improve the performance of the BERT-based classification approaches. They learn on a bigger dataset consisting of user reviews and StackOverflow answers. Thereby, the approach can improve the performance on the PNFR dataset in non-functional requirements subclass classification significantly.

Dell'Anna et al. [33] evaluate the results of available classification methods for detecting functional and quality aspects (NoRBERT [16], Kurtanović and Maalej [13], and Dalpiaz et al. [15]) on an expanded test set with 13 additional datasets. They trained the models on the entire PNFR dataset and tested their performance in terms of precision, recall, and F_1 on the 13 new datasets. On unseen data, NoRBERT achieves the best performance, with an F_1-score of 79% for functional aspects and 71% for quality aspects. However, they could demonstrate that the approach by Dalpiaz et al. [15] exhibits the lowest performance drop between training and testing, making it potentially less prone to overfitting. Nevertheless, on the test set, this approach only achieves a F_1-score of 75% for functional aspects and 62% for quality aspects.

Therefore, NoRBERT remains the current state-of-the-art method for classifying aspects of functional requirements. Due to its promising results on unseen projects and its ease of use without additional preprocessing or manual steps, the method is well-suited for an application in TLR.

B. Traceability Link Recovery

The automatic generation of trace links between requirements and code has been a focus of research since the 1990s.

The first technological breakthrough was achieved through the application of IR techniques. Candidates for TLs are identified based on the textual similarity of the artifacts. Early approaches utilized vector space models (VSMs) [34], [35], latent semantic indexing (LSI) [36] and LDA [37].

Others combine several IR techniques to utilize their strengths. Gethers et al. [38] combine VSMs with probabilistic Jensen and Shannon models (JSs) and relational topic models. Moran et al. [39] utilize a hierarchical Bayesian network to combine multiple IR- and ML-based textual similarities with developer feedback and transitive trace links. Their tool COMET can derive combinations with reasonable performance on unseen projects.

Another direction considers structural information and dependencies within the source code with the rationale that relevant artifacts for a search query can be discovered through the interdependencies within the source code. Panichella et al. [40] demonstrate the advantages of incorporating call and inheritance dependencies in recovering trace links. Kuang et al. [41] additionally utilize data dependencies and show that they offer complimentary information to call dependencies.

TAROT by Gao et al. [42] uses so-called consensual biterms to improve TLR between requirements and code. On the requirements side, TAROT regards two grammatically connected terms of a sentence as biterms. For source code artifacts, any consecutive combination of two terms within an identifier is regarded as a biterm (code comments treated analogously to requirements). The intersection of both biterm sets represents the consensual biterms. The original artifacts are enriched with biterms and the consensual biterms are used to adapt the weights of the IR models based on their frequency and location. The models include VSM, LSI, and JS. In their evaluation, TAROT achieves the best mean average precision (MAP) when using VSM or LSI.

However, all of these techniques have difficulties with linking semantically but not textually similar artifacts. They are not able to bridge the semantic gap between the artifacts, as they rely on textual or syntactical consistency. To address this challenge, recent approaches make use of word embeddings and/or ML. They combine recurrent neural networks (RNNs) and word embeddings [43], feed-forward neural networks and cluster-pair rank models [44], learn to rank word embedding results [45], employ active learning [46], or self-attention [47]. However, these approaches use initial trace links of the projects to train their models and therefore tackle a different kind of TLR problem than this work.

So far, the potential of automatically filtering parts of the requirements was not assessed in TLR research.

IV. Research Design

To assess the potential of requirements classification as a filter on requirements-to-code TLR, we combine the two approaches NoRBERT [16], [18] and FTLR [9]. Therefore, we have to assess what to classify and how to use the results.

A. Classification

There are different classification schemas for requirements that we can consider for filtering irrelevant requirement elements, i.e., parts of requirements that are not helpful or even harmful for automated TLR. In the following, we discuss a sensible classification schema to be used for filtering requirement elements for TLR.

Simply classifying requirements into functional and non-functional requirements can already be instrumental in determining relevant elements. Although a clear-cut separation between functional and non-functional requirements remains a topic of debate [48]–[50], it is still a reason to filter out (parts of) requirements for TLR to source code after recognizing that they mainly describe non-functional properties.

Non-functional requirements, sometimes referred to as soft goals or quality constraints [51], [52], primarily describe quality characteristics of a system and lack a direct functional counterpart. As such, non-functional requirements cannot be directly traced to these non-functional requirements. Thus, they can be filtered for automated TLR.

However, the discussion around separating functional and non-functional requirements revolves around the idea that non-functional requirements can still influence design decisions that manifest in system functionalities. For instance, a non-functional requirement like "The system should support multiple languages" might indirectly lead to the implementation of language selection functionality. Consequently, several researchers advocate for an overlapping classification of requirements containing both functional and non-functional aspects rather than a strict separation [15], [50], [53]. In this classification variant, it is still possible to identify requirements exclusively focusing on non-functional aspects that are considered irrelevant for TLR.

Besides this very coarse classification of requirements, there is often a refinement of classes, particularly for non-functional requirements. These classes are also frequently categorized into further subclasses [12]. This kind of refinement offers a much more precise understanding of the non-functional aspects. However, this type of information is largely irrelevant for TLR. Still, it might be possible to derive probabilities for these subclasses to express the likelihood of implicit functionality being included in these requirements, for instance, Appearance, Security, and Usability might include more functional aspects compared to Legal or Availability. Yet, there is no reliable evidence to make such assertions and, consequently, determining the subclasses of non-functional requirements does not offer advantages for automated TLR.

A more refined classification of functional requirements, on the other hand, plays a smaller role in requirements engineering. One reason for this might be that targeting aspects of non-functional requirements was considered more important, as functional requirements are less frequently overlooked or gathered too late. However, a refined classification of functional requirements could help preprocess automated TLR. Glinz [48] categorizes functional requirements based on their

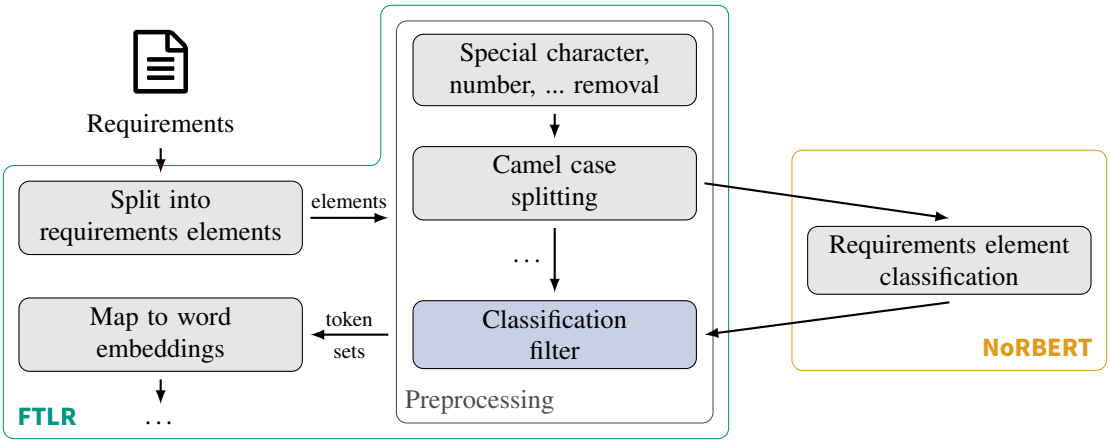

Fig. 4. Integration of NoRBERT's classification results into FTLR

described concerns into Function, Data, Stimuli, Reactions, and Behavior. Hey et al. [16] follow Glinz's classification and provide a labeled dataset based on the functional requirements in the PNFR dataset for the classes Function, Data and Behavior. They consider Reactions and Stimuli as a type of behavior of the system and, thus, aggregate these types. To utilize these classes for automated TLR, one has to be able to determine which parts of the code correspond to the respective concerns. This requires a classification on the source code part as well, which is currently not available. For this reason, we only assess the performance in detecting these concerns and leave the analysis of their impact on TLR to future work.

In addition to these concern-based subcategories of functional requirements, the introductory example in Figure 1 shows another type of potentially irrelevant parts of requirements. Use case descriptions often contain statements that merely describe the behavior of the user who drives the story of the use case. This user behavior is not represented in the source code. The statement "Fill in the form..." is a user interaction that happens outside the system and, thus, should be filtered as well. Therefore, we additionally consider the category of user-related requirements elements:

User-Related (UR): Descriptions of the user's behavior or functionalities attributable to the user within the system.

This leaves us with the following classes for the classification of requirements elements: Elements containing functional aspects (**F**), **Function** concerns, **Data** concerns, **Behavior** concerns and user-related requirements elements (**UR**).

B. Integration of Classification in FTLR

The results of FTLR (cf. [9]) have demonstrated that a fine-grained approach can be advantageous for TLR. Mapping requirements elements to source code elements followed by aggregation yields superior results to mapping the entire artifacts at once. Additionally, utilizing use case description templates for filtering irrelevant requirements elements improves the mapping quality. These results suggest that using information from requirements classification to filter parts of requirements can also improve the performance of TLR.

So far, requirements classification methods classify at the artifact level. This means they classify entire requirements (cf. Section III-A). However, classifying whole use case descriptions, for example, would be too coarse to effectively filter information. The information on (ir-) relevant parts has to be available at the level of requirements elements to effectively filter them. Accordingly, we want to provide information about relevant parts through fine-grained requirements classification, i.e., classification of the distinct elements of the requirements instead of requirements as a whole. The existing classification approaches can be easily adapted to classify requirements elements, as they are mostly trained to classify requirements with one or a few sentences.

FTLR processes input by splitting requirements according to their sentences and use case descriptions following a template into their sections. The section describing the flow of events is further split according to its sentences, as it typically contains multiple steps. FTLR afterward preprocesses the resulting elements to prepare them for representation by the word embeddings (cf. Section II-B). To utilize pre-trained LLM-based classification approaches, such as NoRBERT, which have been trained on complete sentences, we solely apply the initial two preprocessing steps of FTLR before classification: removing special characters, numbers, and hyperlinks, as well as camel case splitting.

We pass these preprocessed requirements elements to the binary NoRBERT classifiers. The classifiers determine whether a requirements element belongs to one of the classes identified in Section IV-A. The classification results are then returned to FTLR and are used to exclude elements from consideration for TLR. The classification of requirements elements and the subsequent filtering, thus, can be considered a new preprocessing step within FTLR. This process is illustrated in Figure 4.

We have three types of filters: The first type of filter is for functional aspects (`F`) and removes all requirements elements that do not contain functional aspects according to the classification. The second type of filter, the user-related elements filter (`UR`), removes requirements elements that involve user-related

behavior. The third filter, the F+UR filter, uses the other two filters to remove all elements that either contain no functional aspects or contain user-related descriptions.

It is important to note that an element lacking functional aspects does not necessarily qualify as a non-functional requirement according to Glinz' definition [54]. It merely contains natural language text that lacks functional aspects. Since these kinds of elements are common in use case descriptions, classifying whether an element contains functional aspects is preferable over determining if it entails quality aspects or is a non-functional requirement.

V. Empirical Study on Classification

In this first experiment, we assess the classification performance of NoRBERT on requirements elements from projects commonly used as benchmarks in requirements-to-code TLR. The experiments aim to answer the following questions:

RQ1: *Can the performance of a LLM-based classification approach on unseen projects be transferred from whole requirements to requirements elements?*

We regard the performance on unseen projects as particularly relevant, as the classification should be used as a preprocessing step on any given project and, thus, no training data for the specific project might be available beforehand.

RQ2: *Which NoRBERT configurations perform best on the TLR datasets?*

The authors of NoRBERT reported that depending on the task and classes, a different set of hyperparameters performed best. As a high quality of the classification results is needed, if they are to be used as a filter in FTLR, the best performing configuration should be used per class.

To be able to answer these questions, we manually labeled the requirements elements of eTour, iTrust, SMOS, eAnci, and LibEST regarding the classes *functional aspects* (F), *user-related* (UR), *Function*, *Data*, and *Behavior*. The projects are commonly used as benchmarks in requirements-to-code TLR. The first four projects are provided by the *Center of Excellence for Software & Systems Traceability* (CoEST) [55] and LibEST was provided by Moran et al. [39]. eTour, eAnci and SMOS comprise use case descriptions following templates that at least determine the name, description, pre-, and post-conditions as well as the flow of events. iTrust only includes the flow of events of use case descriptions and LibEST has textual requirements. However, only the requirements of eTour, iTrust and LibEST are written in English, SMOS and eAnci have Italian requirements. As NoRBERT can only classify English inputs due to the available data (PNFR and the pretraining of the BERT model), the Italian projects cannot be used as is. As such, we automatically translate their requirements using the Google Translate API.

To have a gold standard of labeled requirements elements for these projects, we manually labeled them. The labeling process was carried out independently by one of the authors and a master's student in computer science. This resulted in a very high agreement for the classes of functional aspects (F), Behavior, and user-related elements (UR), with inter-annotator agreements of 0.897 (F), 0.887 (Behavior), and 0.905 (UR) using Krippendorff's α [56]. Similarly, the agreement for the classes of Function and Data with α values of 0.742 and 0.785, respectively, are clearly above the lower threshold for acceptable agreement of 0.66 [57]. Discrepancies were resolved through discussions to determine a consistent solution. Table II shows the performance of the human annotators regarding the derived gold standard solution, giving insights into the human achievable performance. The resulting gold standard is available on zenodo [58].

TABLE I
Distribution of the Classes Over the Requirements Elements (ReqE) That Are in the Requirements (Req) of the Datasets

Dataset	Req	ReqE	F	Fun.	Beh.	Data	UR
eTour	58	571	457	299	342	186	279
iTrust	131	336	311	179	164	169	205
SMOS	67	522	386	243	293	276	251
eAnci	139	1,290	874	570	561	328	672
LibEST	52	565	404	209	290	205	5
Total	447	3,284	2,432	1,500	1,650	1,164	1,412
+ PNFR	691	3,908	2,741	1,707	1,763	1,221	1,412

TABLE II
Average Performance in Precision, Recall, F_1, and Accuracy of Human Annotators Regarding the Resulting Gold Standard

Class	Precision	Recall	F_1	Accuracy
F	.958	.991	.975	.962
UR	.911	.987	.947	.953
Function	.839	.890	.864	.871
Behavior	.946	.942	.944	.944
Data	.870	.851	.861	.902

The distribution of classes across the projects is illustrated in Table I. By splitting requirements (Req) into requirements elements (ReqE), there is a substantial increase in data points for classification, totaling 3,284 elements. Particularly, the 139 use case descriptions in the eAnci project contain 1,290 elements, constituting over a third of all elements. However, only 874 of eAnci's elements actually contain functional aspects. Hence, 416 out of these 1,290 elements are texts that need to be filtered. The other datasets generate similar-sized sets of elements with around 550 elements. Only the iTrust dataset, which exclusively includes the flow of events descriptions as requirements, yields merely 336 elements for the original 131 requirements. However, all except for 25 of these elements also include functional aspects. Therefore, the filtering of requirements elements can only partially reduce the input for the TLR in iTrust.

The two classes, Function and Behavior, show a relatively high number of instances across all datasets, with eTour, SMOS, and LibEST containing more behavior descriptions

than function descriptions. For the networking library LibEST, this can be attributed to many sentences explaining the network stack's interaction rather than explicitly describing offered functions. Additionally, most datasets include some elements that describe both the function itself and its associated behavior. The Data class with only 1,164 instances overall is less frequently present compared to the other concerns, much like in the PNFR-based dataset of Hey et al. [16]. In the datasets with use case descriptions, around 50% of the elements are user-related elements, whereas only 5 of 565 elements in the non-user-centric requirements of LibEST are user-related.

A. Experimental Setup

The classification should apply to any given project without retraining as it should be used as a preprocessing step for TLR. Consequently, NoRBERT's performance in a scenario without knowledge of the project to classify should be assessed. Therefore, in our evaluation, we use a setup that resembles a form of leave-one-project-out (loPo) cross-validation strategy, except that we do not measure the performance on the PNFR projects. We alternately consider each of the projects once as the unseen project, while NoRBERT is trained on the other projects plus the whole PNFR dataset. This way, NoRBERT has the maximum amount of training data for each project without including the project itself in the test. To train the user-related class, there is no information available from the PNFR dataset, as Hey et al. [16] did not consider this class. Therefore, training for UR is only conducted on the remaining four traceability projects, respectively.

To evaluate the overall performance, we use accuracy and F_1-score as metrics on each project and also consider the average both without weighting ($\varnothing$) and weighted by the respective occurrences in the project ($\varnothing_w$). The weighted representation is less sensitive to deviating results in projects with very few instances of a class (e.g., the five UR instances in LibEST). Conversely, the unweighted average is less influenced by deviating (potentially very good) results in projects with a high number of instances of a class (e.g., eAnci).

We measure both accuracy and F_1-score, as together they provide a more comprehensive overview of binary classification results. Accuracy displays the overall performance in both identifying the occurrences (true positives) and non-occurrences (true negatives) of a certain class in the dataset. However, accuracy is prone to imbalanced datasets, for example with a dataset with only a few occurrences of a certain class, where a high accuracy can be achieved by only predicting no occurrence. Therefore, we also provide precision, recall, and F_1-score that focus on the occurrences only. We choose F_1-score instead of a different F-measure, as for our following task of filtering irrelevant requirements elements both recall and precision are equally important. When valuing recall higher than precision, a lower precision could result in not being able to filter any requirements elements as too many are considered for example functional. If we value precision higher than recall, a lower recall may result in actually relevant elements being filtered as they are regarded as, e.g., having no functional aspects.

TABLE III
ACCURACY OF NORBERT USING THE BERT-LARGE MODEL AND 16 EPOCHS WITH AND WITHOUT OVERSAMPLING (OS).

Class	OS	eTour	iTrust	SMOS	eAnci	LibEST	$\varnothing$	$\varnothing_w$
F	✖	.912	.943	.738	.838	.696	.825	.826
	✔	.907	.958	.753	.860	.710	**.838**	**.840**
UR	✖	.722	.759	.784	.922	.591	.756	.833
	✔	.739	.759	.768	.933	.596	**.759**	**.839**
Func.	✖	.816	.696	.672	.705	.600	**.698**	**.706**
	✔	.774	.676	.661	.715	.653	.696	.705
Beh.	✖	.778	.720	.912	.779	.558	**.749**	**.758**
	✔	.779	.720	.770	.750	.565	.717	.724
Data	✖	.743	.777	.634	.779	.612	**.709**	**.709**
	✔	.823	.774	.711	.792	.363	.693	.700

As the classes of our classification can overlap and, thus, the task at hand is a multi-label classification, we use a binary classifier for each class. The authors of NoRBERT reported that epoch numbers of 10 or 16 with and without oversampling (OS) performed best on the PNFR dataset. We also use these configurations in combination with either the $\text{BERT}_{\texttt{BASE}}$ or the $\text{BERT}_{\texttt{LARGE}}$ model. Due to space limitations, we only present the best-performing configuration consisting of $\text{BERT}_{\texttt{LARGE}}$ with 16 epochs. Further, results can be found in our supplementary material [17].

B. Classification Results

Table III shows the resulting accuracy and Table IV the precision, recall, and F_1-score across the projects and classes. For the two classes selected for filtering in FTLR, F and UR, the model with oversampling performs best with a weighted accuracy of 84%. To identify functional aspects (F) in eTour and iTrust, the model even achieves accuracies and F_1-scores of over 90%. For this class, the unweighted average is nearly equal to the weighted, whereas for identifying user-related elements (UR) the accuracy declines to 75.9%. That can be explained by the far lower performance on the LibEST dataset (59.6%), where only 5 of 565 requirements elements are user-related. A closer look at the precision and recall of the approach in Table IV shows that the classifier can identify 4 of these 5 elements (recall of 80%) but also introduces a lot of false positives resulting in a low precision of 1.7%. As the classifier in this setting trains only on use case descriptions but LibEST contains regular requirements, it is not able to generalize well enough for the user-related class. For the other classes, this decline cannot be observed. In this special case with high recall and few user-related requirements, a semi-automated approach with manual vetting could easily fix this.

TABLE IV
PRECISION, RECALL, AND F_1-SCORES OF NORBERT USING THE BERT-LARGE MODEL AND 16 EPOCHS WITH AND WITHOUT OVERSAMPLING (OS).

		eTour			iTrust			SMOS			eAnci			LibEST			F_1	
Class	OS	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	$\varnothing$	$\varnothing_w$
F	✖	.901	1.00	.948	.965	.974	.970	.743	.987	.848	.817	.981	.891	.721	.938	.815	.894	.892
	✔	.899	.996	.945	.968	.987	.978	.756	.982	.855	.850	.965	.904	.759	.871	.811	**.898**	**.898**
UR	✖	.962	.448	.611	.903	.678	.774	.825	.697	.756	.940	.909	.924	.017	.800	.033	.620	.808
	✔	.933	.502	.653	.913	.668	.772	.795	.697	.743	.947	.923	.934	.017	.800	.034	**.627**	**.818**
Func.	✖	.790	.883	.834	.785	.592	.675	.597	.914	.722	.686	.614	.648	.466	.550	.504	**.677**	**.680**
	✔	.772	.806	.789	.797	.525	.633	.586	.926	.718	.721	.579	.642	.527	.608	.564	.669	.672
Beh.	✖	.745	.956	.837	.787	.585	.671	.919	.925	.922	.694	.879	.776	.596	.428	.498	.741	**.755**
	✔	.744	.962	.839	.750	.640	.691	.717	.976	.827	.659	.879	.753	.561	.697	.622	**.746**	**.755**
Data	✖	.577	.785	.665	.737	.864	.796	.660	.634	.647	.591	.427	.496	.445	.278	.342	.589	.575
	✔	.701	.796	.746	.751	.822	.785	.870	.533	.661	.623	.463	.531	.363	1.00	.532	**.651**	**.633**

The weighted average of both accuracy and F_1-score shows promising results. The results on both classes are even higher than the ones reported by Hey et al. [18] for classifying functional aspects in a loPo setting on the PNFR dataset (81% weighted F_1-score). However, for the three classes of functional requirements concerns (Function, Data and Behavior) the model without oversampling performs best in accuracy. The results on weighted and unweighted averages do not differ as much as they are more evenly distributed across the projects. The results are lower than for F or UR with 70.6% for Function, 70.9% for Data, and 75.8% for Behavior. In F_1-score, however, oversampling again performs better for the Behavior and Data class. This showcases the weaknesses of accuracy as a metric: it can be dominated by many true negatives. The human annotators achieved the lowest precision (cf. Table II) for Function and Data which can also be observed in the precision results of the classifier. On average, the results in F_1-score and accuracy are 20 percentage points (%pts) lower than the results of the human annotators.

Comparing the results in F_1-score to the results reported by Hey et al. [18] for loPo on the PNFR dataset, only the Function class performs worse (68% vs. 87%). For Behavior, the result is 4%pts higher and for Data even 6%pts higher. This can be explained by an increased number of training examples for these classes. On the PNFR dataset, the data class had only 57 examples. Now the dataset comprises 1,221 examples of the class and, thus, the results can support the authors' claim.

The results show that a LLM-based classification approach can classify requirements elements on TLR datasets with a comparable or even better performance than on whole requirements (RQ1). Regarding RQ2, a configuration can be determined for each class that performs best in weighted and unweighted average accuracy and F_1-score across the projects. For F, UR, Data, and Behavior, this is $\text{BERT}_{\text{LARGE}}$ with 16 epochs and oversampling. For Function, it is $\text{BERT}_{\text{LARGE}}$ with 16 epochs and no sampling.

C. Threats to Validity

In this section, we discuss the threats to validity of this first experiment and the derived conclusions.

External Validity: The loPo evaluation strategy was selected to showcase the generalizability of NoRBERT on requirements elements. However, the projects within the datasets might not necessarily be good representatives of possible projects, posing a threat to the external validity of the results. Additionally, both the PNFR dataset and the traceability datasets are biased representations of reality, either focusing on non-functional or functional requirements. This results in unbalanced datasets, potentially shadowing the results by favoring simple majority class predictions. Moreover, all requirements within the PNFR dataset projects and some traceability datasets were authored by students, not necessarily reflecting industry standards. Therefore, the validity of claims regarding generalizability based on these datasets cannot be guaranteed. Still, these datasets are considered benchmark datasets for requirements classification and for TLR and are widely used to compare different approaches.

Internal Validity: A threat to internal validity arises from creating the gold standard for the traceability datasets. These were created with the approach in mind, posing a risk of bias. Furthermore, the usage of machine translation for the requirements of SMOS and eAnci may introduce another bias. As automatic translation approaches can introduce errors themselves, a classification by NoRBERT can be influenced by those errors. To enable its verification and the reproducibility of the results, the dataset is publicly available [58].

VI. EMPIRICAL STUDY ON THE IMPACT ON TLR

The second experiment of this work aims at assessing the impact of automated classification of requirements elements on TLR. Therefore, the following sections will evaluate the effect of filtering requirements elements based on NoRBERT. We use the best performing NoRBERT configurations according to the

TABLE V
AVERAGED RESULTS OF THE DIFFERENT FTLR VARIANTS WITH THE DIFFERENT FILTERS. EITHER THE ORIGINAL THRESHOLDS (ORG) OR PER PROJECT OPTIMIZED THRESHOLDS (OPT) ARE APPLIED.

Thresh.	Filter				MC			MC+CD		
		P	R	F_1	P	R	F_1	P	R	F_1
ORG	—	.287	.423	.327	.291	.466	.344	.294	.459	.345
	F	.313	.399	.335	.311	.445	**.353**	.315	.436	**.353**
	UR	.373	.312	.302	.363	.334	.317	.367	.330	.318
	F+UR	.389	.302	.299	.365	.319	.308	.368	.315	.308
	F_{gold}	.336	.394	.349	.334	.443	.369	.339	.435	.370
	UR_{gold}	.371	.351	.342	.368	.389	.363	.370	.383	.360
	$F+UR_{gold}$	.379	.342	.342	.374	.385	.365	.377	.380	.363
OPT	—	.290	.532	.372	.316	.506	.378	.298	.527	.375
	F	.318	.509	**.388**	.337	.466	**.388**	.328	.473	.386
	UR	.311	.487	.371	.332	.452	.379	.329	.453	.378
	F+UR	.315	.472	.369	.329	.446	.375	.328	.445	.375
	F_{gold}	.359	.474	.405	.342	.515	.404	.344	.506	.402
	UR_{gold}	.346	.480	.394	.359	.462	.395	.364	.456	.392
	$F+UR_{gold}$	.362	.470	.396	.366	.444	.396	.358	.461	.394

TABLE VI
AVERAGED RESULTS OF THE DIFFERENT FTLR VARIANTS WITH THE DIFFERENT FILTERS. EITHER THE ORIGINAL THRESHOLDS (ORG) OR PER PROJECT OPTIMIZED THRESHOLDS (OPT) ARE APPLIED.

Thresh.	Filter				MC			MC+CD		
		P	R	F_1	P	R	F_1	P	R	F_1
ORG	F	.313	.399	.335	.311	.445	.353	.315	.436	.353
	UCT	.331	.399	.346	.330	.445	.366	.335	.436	.366
	UCT+F	.337	.388	.347	.337	.435	**.367**	.341	.426	**.367**
	F_{gold}	.336	.394	.349	.334	.443	.369	.339	.435	.370
	$UCT+F_{gold}$	.335	.389	.346	.334	.439	.367	.338	.430	.367
OPT	F	.318	.509	.388	.337	.466	.388	.328	.473	.386
	UCT	.349	.500	.403	.353	.491	.403	.342	.503	.401
	UCT+F	.351	.499	**.407**	.366	.458	.405	.356	.466	.403
	F_{gold}	.359	.474	.405	.342	.515	.404	.344	.506	.402
	$UCT+F_{gold}$	.348	.501	.407	.345	.506	.404	.346	.499	.402

results in Section V-B to generate the classification results on the datasets eTour, iTrust, SMOS, eAnci, and LibEST. However, FTLR's code analysis features can only be applied to the Java part [9] and, as such, we need to exclude the JSP artifacts that are part of the iTrust dataset. The classification results are generated according to the loPo strategy described in Section V-A to provide a realistic scenario for using NoRBERT combined with FTLR.

We again use precision, recall and F_1-score as metrics to compare the results. We choose F_1-score as the deciding measure, as for the goal to fully automate TLR both precision and recall have to be high [6]. Additionally, F_1-score is one of the standard metrics in TLR research [7] allowing us to compare the results to existing approaches. If one weakens the goal to semi-automatic TLR recall should be valued higher to prevent missing links. Therefore, we additionally provide F_2-score and mean average precision (MAP) for comparison with state-of-the-art approaches.

The results achieved by NoRBERT on the two classes used as filters (F and UR) are promising but not perfect. Consequently, there is a possibility that relevant parts of the input may be erroneously removed due to misclassifications. To assess the upper limits of the performance, i.e., the results achievable with an optimal classification, FTLR is also executed with the gold standard results for the classification (marked with $[F|UR]_{gold}$).

We first compare the performance of FTLR with requirements element filtering with FTLR without any element filter and without the UCT option. As UCT itself represents a filter that operates based on use case template elements, this experimental setup shows the influence of automated requirements element filtering on unfiltered inputs. The introduction of requirements element filters was intended for use on requirements that do not follow a template. Additionally, a requirements element filter can be employed without explicit knowledge of the template elements and their naming even when use case descriptions adhere to a template. To quantify this effect and to verify whether the performance of both filter types is comparable, we then compare the performance of the requirements element filters with the performance of filtering based on use case templates (UCT). Finally, we assess whether a combination of the two filter variants achieves a performance improvement over their individual applications. Therefore, this chapter addresses the following research questions:

RQ3: *What performance does FTLR achieve with an automated requirements element filter compared to FTLR without any requirement filter?*

RQ4: *What is the upper limit of the performance that FTLR with a requirements element filter can achieve on the benchmark datasets?*

RQ5: *Is the performance of FTLR with an automated requirements element filter comparable to that based on filtering of use case template elements (UCT)?*

RQ6: *Can a combination of an automated element filter and filtering based on use case template elements improve the performance of FTLR compared to the individual variants?*

Additionally, we assess the impact in comparison to other TLR approaches by providing a comparison with several state-of-the-art and baseline approaches:

RQ7: *How does FTLR with an automated requirements element filter fair in comparison to state-of-the-art and baseline TLR approaches?*

A. Comparison to FTLR without Filters

First, we measure FTLR's performance regarding precision, recall and F_1-score with and without an automated

TABLE VII
COMPARISON TO RELATED WORK IN REQUIREMENTS-TO-CODE TLR USING OPTIMIZED THRESHOLDS PER PROJECT

		eTour			iTrust w/o JSP			SMOS			eAnci			Average		
Approach	Filter	F_1	F_2	MAP	F_1	F_2	MAP	F_1	F_2	MAP	F_1	F_2	MAP	F_1	F_2	MAP
VSM		.483	.448	.464	.217	.223	.242	.422	.417	.451	.248	.252	.457	.343	.335	.404
LSI		.453	.453	.449	.253	.254	.265	.422	.427	.468	.217	.237	.390	.336	.343	.393
$TAROT_{VSM}$		.402	.503	.384	.224	.226	.240	.418	.416	.449	.258	.275	.466	.326	.355	.385
$TAROT_{LSI}$		.403	.510	.407	.217	**.276**	.252	**.461**	**.517**	**.496**	.226	.264	.403	.327	.392	.390
$COMET_{MAP}$		.437	.455	.475	**.282**	.249	.263	.276	.458	.294	—	—	—	(.332)	(.387)	(.344)
FTLR +MC	+UCT	.517	.548	.486	.253	.255	**.354**	.419	.446	.451	.276	.290	.549	.366	.385	**.460**
FTLR +MC	+F	.442	.519	.369	.261	.252	.348	.410	.462	.424	.271	**.349**	**.568**	.346	.396	.427
FTLR	+UCT +F	**.548**	**.576**	**.533**	.238	.240	.283	.409	.500	.438	**.289**	.306	.533	**.371**	**.406**	.447
FTLR +MC+CD	+UCT +F	.517	.543	.486	.250	.273	.333	.418	.457	.450	.275	.304	.541	.362	.394	.453

requirements element filter. Table V compares the results of FTLR averaged across the five datasets. We compare FTLR without a filter (—), with the functional aspects filter (F), with the user-related filter (UR) and a combination of both filters (F+UR). As the authors of FTLR provide a fixed set of thresholds (ORG) and the possibility to optimize the thresholds per project (OPT) based on a gold standard, we provide both results. The former measures FTLR's performance when applied in practice without initial links, the latter provides the upper bound of performance with the approach on a specific project. Additionally, we assess the performance when including method comments (MC) and method comments together with call dependencies (MC+CD).

The results show that the best averaged F_1-score with automated requirements element classification (bold typed values) is achieved by only applying the F filter, regardless of the threshold combination used. With ORG thresholds, FTLR using the F filter achieves 35.3% F_1-score with both FTLR variants including method comments (MC and MC+CD). This is 0.8%pts higher than the best FTLR variant without a filter. With optimized thresholds, the difference increases further from 37.8% to 38.8% (both with MC). This improvement is statistically significant at the 0.05 level using a Wilcoxon signed-rank test (RQ3).

On average, the UR filter performs worse than FTLR without any filter, mainly due to a heavy decline in the recall. However, our detailed results (cf. [17]) show that for the projects eTour and eAnci the improvement in precision outweighs the decline in recall, resulting in an improved F_1-score. With optimized thresholds, the results are similar, although the decline in performance is smaller.

To provide insights for RQ4, Table V also includes the results with requirements element classification from the gold standard ($_{gold}$). The highlighted best-performing results are again only achieved with the F filter. For the ORG thresholds, the averaged F_1-score improves to 37% (+2.5%pts compared to no filter). With optimized thresholds, the best averaged F_1-score is 40.5% when not using method comments (MC) and call dependencies (CD). These results show the upper bound of FTLR's performance with these filters on these datasets.

B. Comparison to UCT Filter

In this part, we tackle RQ5 and check whether the results with the automated requirements elements filter are comparable to the ones of a manually provided use case template filter. To do so, we compare FTLR's performance with the F filter to the one with the fixed use case template filter (UCT).

Table VI shows the averaged results across the five TLR projects. Both with ORG and OPT thresholds, FTLR with UCT performs slightly better than FTLR with F filter, mainly due to its better precision. However, the difference is negligible as it stays below 1.5%pts. It is important to note that the UCT filter only removes the use case fields describing actors, pre- and postcondition, and quality requirements. Thus, it is more conservative than the F filter. Additionally, the UCT filter is only applicable to the eTour, eAnci, and SMOS projects, as only these comprise use case descriptions following templates. On iTrust, the F_1-score using FTLR ORG UCT (18.3%) is 2%pts lower than with F (20.3%).

The results of combining UCT with the F filter show an improvement over applying only UCT, especially with optimized thresholds. This indicates that the F filter can filter further irrelevant elements and that a combination of both filters is beneficial (RQ6). The overall best performance in average F_1-score is achieved with 40.7% by FTLR with UCT and F filters without MC or CD. The result is even similar to the one with gold standard filter results.

C. Comparison to Related Work

For improved interpretation of the results achieved by FTLR with requirement filters, we compare them to state-of-the-art and baseline approaches for requirements-to-code TLR. We limit the comparison to approaches that do not require initial trace links as training data and, thus, are targeting the same task of unsupervised TLR as FTLR. We use COMET [39] and TAROT [42] (cf. Section III-B) as competing approaches. As the approaches only output ranked lists per source artifact and do not define a fixed threshold, we calculate the optimized

F_1-score per project as we did for FTLR and only compare to FTLR's `OPT` results. Thus, the comparison reflects the upper boundary of the tools' performance.

For COMET, we use the ranked lists as provided by the authors to calculate the MAP and the per-project optimized F_1-score. We cannot provide results for eAnci because it was not part of the original COMET evaluation. As the replication package of TAROT provides its source code, we can apply TAROT to our datasets. Additionally, we provide the results of two baseline approaches using VSM and LSI as provided in TAROT's replication package. As comparison projects, we use eTour, iTrust without JSP artifacts, SMOS, and eAnci because TAROT can only be applied to Java projects. For FTLR, we provide the results of the best-performing combinations that use either `UCT` and/or `F` filter.

Table VII gives an overview of the results. The findings indicate that, on average, FTLR outperforms all comparison approaches irrespective when using `F` as shown by higher F_1-score, F_2-score, and MAP values. In F_1-score, FTLR with `UCT+F` achieves the best performance with 37.1%, which is 2.8%pts higher than the runner-up approach VSM. If we weigh recall higher (F_2-score), the same FTLR variant performs best with 40.6%. In F_2-score the runner-up approach is TAROT with LSI with 39.2%, which is 1.2%pts lower than FTLR's. Detailed results like precision and recall of the approaches can be found in our supplementary material [17]. In MAP, the difference is even higher, surpassing the others by 4.3%pts for FTLR +`UCT+F` and by 5.6%pts with FTLR +`UCT`.

However, on iTrust, COMET yields a higher F_1-score and $\text{TAROT}_{\text{LSI}}$ a higher F_2-score, but their MAP are still 1.1%pts or 2%pts lower than FTLR's. On SMOS, the VSM- and LSI-based approaches outperform COMET and FTLR, with TAROT using LSI performing best. As $\text{TAROT}_{\text{LSI}}$ performs worse than FTLR and COMET on the other projects, this result might be specific for the SMOS dataset. This dataset comprises the highest number of interconnected artifacts of all the projects with 16% of the possible interconnection between artifacts being actual trace links. Notably, the baseline approaches using VSM and LSI perform better than the versions using consensual biterms, although these biterms are the key feature of TAROT. This contradicts the results of the authors of TAROT on other datasets.

In summary, the usage of requirements filter in FTLR has shown to be beneficial when comparing it to state-of-the-art approaches (RQ7). In future work, the application of these filters to other TLR approaches needs to be examined to assess whether this outcome can be generalized to other approaches.

D. Threats to Validity

External validity is the first and arguably the most significant threat to the validity of the insights of this experiment. The experiments were conducted on a limited set of projects, predominantly originating from academic or student projects. Therefore, there is a possibility that the obtained results are not representative of other projects, particularly in industry, and, as a result, the findings may not apply to other projects. This risk is shared by all existing approaches for automatic TLR since fully representative datasets with an established gold standard for trace links are non-existent.

Still, the selected datasets contain projects that vary in size and language and cover different domains. The datasets are widely used as benchmarks in the traceability research community and are generally accepted for comparison matters.

VII. Conclusion

In conclusion, this work investigated the potential of requirements classification for traceability link recovery (TLR) between requirements and code. Therefore, we combine the requirements classification approach NoRBERT with the approach FTLR for automated TLR. NoRBERT was applied to a new dataset for classifying requirements elements in traceability benchmark projects. The results of NoRBERT's classification are then used to filter irrelevant parts of the requirements before conducting the traceability link recovery with FTLR. We performed two empirical studies to assess the potential of the proposed approach.

In the first study, we evaluated NoRBERT's performance in identifying functional and user-related aspects as well as function, behavior, and data concerns in the requirements elements of the new dataset. Functional and user-related aspects were identified with a promising weighted F_1-score of 84%. Further, NoRBERT achieved F_1-scores of more than 70% for each of the three functional concerns.

The second study used the classification results to assess the impact on TLR. We were able to show that an automated requirements filter based on functional aspects can significantly improve FTLR's performance in our evaluation. Additionally, the results show that the automated filter can provide comparable results to a filter based on use case templates with manual labels. Combining both types of filters could improve FTLR's performance from 37.8% to 40.7% F_1-score. In comparison to state-of-the-art TLR approaches, FTLR with these filters performed best, both in F_1 and MAP.

Our studies were able to show, that an automated classification of TLR relevant aspects in requirements is possible. With this classification, we can filter irrelevant information. We have shown that this filtering can be advantageous for TLR and increase the usability for practitioners. Due to automation, enhancing existing approaches or including the classification as a preprocessing step when researching new approaches is straightforward. Consequently, in future work, we plan to integrate the classification results in further TLR approaches to assess the generalizability of our insights to other approaches. Additionally, we will investigate the potential of the function, data, and behavior concerns for directing the mapping to the most likely elements in the code.

Acknowledgements

This work was supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF). This work was also supported by funding from the topic Engineering Secure Systems of the HGF and supported by KASTEL Security Research Labs, Karlsruhe.

References

[1] J. Cleland-Huang, O. Gotel, A. Zisman *et al.*, *Software and Systems Traceability*. Springer, 2012, vol. 2. DOI: 10.1007/978-1-4471-2239-5

[2] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic, "The Grand Challenge of Traceability (v1.0)," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. London: Springer, 2012, pp. 343–409. DOI: 10.1007/978-1-4471-2239-5_16

[3] RTCA/EUROCAE, "DO-178B/ED-12B: Software considerations in airborne systems and equipment certification," 2000.

[4] ECSS, "ECSS-E-40C: Principles and requirements applicable to space software engineering," 2009.

[5] L. Rierson, *Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance*. CRC Press, Jan. 2013.

[6] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods," *IEEE Trans. Softw. Eng.*, vol. 32, no. 1, pp. 4–19, Jan. 2006. DOI: 10.1109/TSE.2006.3

[7] Y. Shin, J. H. Hayes, and J. Cleland-Huang, "Guidelines for Benchmarking Automated Software Traceability Techniques," in *2015 IEEE/ACM 8th International Symposium on Software and Systems Traceability*, May 2015, pp. 61–67. DOI: 10.1109/SST.2015.13

[8] G. Antoniol, J. Cleland-Huang, J. H. Hayes, and M. Vierhauser, "Grand Challenges of Traceability: The Next Ten Years," *arXiv:1710.03129 [cs]*, Oct. 2017. [Online]. Available: http://arxiv.org/abs/1710.03129

[9] T. Hey, F. Chen, S. Weigelt, and W. F. Tichy, "Improving Traceability Link Recovery Using Fine-grained Requirements-to-Code Relations," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2021, pp. 12–22. DOI: 10.1109/ICSME52107.2021.00008

[10] J. Slankas and L. Williams, "Automated extraction of non-functional requirements in available documentation," in *2013 1st International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, May 2013, pp. 9–16. DOI: 10.1109/NAturaLiSE.2013.6611715

[11] J. Winkler and A. Vogelsang, "Automatic Classification of Requirements Based on Convolutional Neural Networks," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, Sep. 2016, pp. 39–45. DOI: 10.1109/REW.2016.021

[12] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "The Detection and Classification of Non-Functional Requirements with Application to Early Aspects," in *14th IEEE International Requirements Engineering Conference (RE'06)*, Sep. 2006, pp. 39–48. DOI: 10.1109/RE.2006.65

[13] Z. Kurtanović and W. Maalej, "Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Sep. 2017, pp. 490–495. DOI: 10.1109/RE.2017.82

[14] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What Works Better? A Study of Classifying Requirements," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Sep. 2017, pp. 496–501. DOI: 10.1109/RE.2017.36

[15] F. Dalpiaz, D. Dell'Anna, F. B. Aydemir, and S. Çevikol, "Requirements Classification with Interpretable Machine Learning and Dependency Parsing," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*, Sep. 2019, pp. 142–152. DOI: 10.1109/RE.2019.00025

[16] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, "NoRBERT: Transfer Learning for Requirements Classification," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, Aug. 2020, pp. 169–179. DOI: 10.1109/RE48521.2020.00028

[17] T. Hey, J. Keim, and S. Corallo, "Supplementary Material of "Requirements Classification for Traceability Link Recovery"," Zenodo, 2024. DOI: 10.5281/zenodo.10990762

[18] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, "NoRBERT: Transfer learning for requirements classification," Karlsruhe Institute of Technology (KIT), Tech. Rep., 2022. DOI: 10.5445/IR/1000150464

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423

[20] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 19–27. DOI: 10.1109/ICCV.2015.11

[21] Jane Cleland-Huang, S. Mazrouee, H. Liguo, and D. Port, "Nfr," Mar. 2007. DOI: 10.5281/zenodo.268542

[22] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning Word Vectors for 157 Languages," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. [Online]. Available: https://aclanthology.org/L18-1550

[23] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, "From Word Embeddings To Document Distances," in *International Conference on Machine Learning*. PMLR, Jun. 2015, pp. 957–966.

[24] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requir. Eng.*, vol. 12, no. 2, pp. 103–120, May 2007. DOI: 10.1007/s00766-007-0045-1

[25] J. Sayyad Shirabad and T. Menzies, "The PROMISE repository of software engineering databases." School of Information Technology and Engineering, University of Ottawa, Canada, 2005. [Online]. Available: http://promise.site.uottawa.ca/SERepository

[26] I. Hussain, L. Kosseim, and O. Ormandjieva, "Using Linguistic Knowledge to Classify Non-functional Requirements in SRS documents," in *Proceedings of the 13th International Conference on Natural Language and Information Systems: Applications of Natural Language to Information Systems*, ser. NLDB '08. London, UK: Springer-Verlag, Jun. 2008, pp. 287–298. DOI: 10.1007/978-3-540-69858-6_28

[27] M. Rahimi, M. Mirakhorli, and J. Cleland-Huang, "Automated extraction and visualization of quality concerns from requirements specifications," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, Aug. 2014, pp. 253–262. DOI: 10.1109/RE.2014.6912267

[28] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, "Towards Supporting Software Engineering Using Deep Learning: A Case of Software Requirements Classification," in *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*, Oct. 2017, pp. 116–120. DOI: 10.1109/CONISOFT.2017.00021

[29] A. Dekhtyar and V. Fong, "RE Data Challenge: Requirements Identification with Word2Vec and TensorFlow," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, Sep. 2017, pp. 484–489. DOI: 10.1109/RE.2017.26

[30] S. Amasaki and P. Leelaprute, "The Effects of Vectorization Methods on Non-Functional Requirements Classification," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug. 2018, pp. 175–182. DOI: 10.1109/SEAA.2018.00036

[31] G. Li, C. Zheng, M. Li, and H. Wang, "Automatic Requirements Classification Based on Graph Attention Network," *IEEE Access*, vol. 10, pp. 30 080–30 090, 2022. DOI: 10.1109/ACCESS.2022.3159238

[32] X. Luo, Y. Xue, Z. Xing, and J. Sun, "PRCBERT: Prompt Learning for Requirement Classification using BERT-based Pretrained Language Models," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, Jan. 2023, pp. 1–13. [Online]. Available: https://dl.acm.org/doi/10.1145/3551349.3560417. DOI: 10.1145/3551349.3560417

[33] D. Dell'Anna, F. B. Aydemir, and F. Dalpiaz, "Evaluating classifiers in SE research: the ECSER pipeline and two replication studies," *Empir Software Eng*, vol. 28, no. 1, Nov. 2022. DOI: 10.1007/s10664-022-10243-1

[34] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, Oct. 2002. DOI: 10.1109/TSE.2002.1041053

[35] A. Mahmoud, "An information theoretic approach for extracting and tracing non-functional requirements," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*, Aug. 2015, pp. 36–45. DOI: 10.1109/RE.2015.7320406

[36] A. Marcus and J. I. Maletic, "Recovering Documentation-to-source-code Traceability Links Using Latent Semantic Indexing," in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 125–135. [Online]. Available: http://dl.acm.org/citation.cfm?id=776816.776832

[37] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, May 2010, pp. 95–104. DOI: 10.1145/1806799.1806817

[38] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, Sep. 2011, pp. 133–142. DOI: 10.1109/ICSM.2011.6080780

[39] K. Moran, D. N. Palacio, C. Bernal-Cárdenas, D. McCrystal, D. Poshyvanyk, C. Shenefiel, and J. Johnson, "Improving the effectiveness of traceability link recovery using hierarchical bayesian networks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 873–885. DOI: 10.1145/3377811.3380418

[40] A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "When and How Using Structural Information to Improve IR-Based Traceability Recovery," in *2013 17th European Conference on Software Maintenance and Reengineering*, Mar. 2013, pp. 199–208. DOI: 10.1109/CSMR.2013.29

[41] H. Kuang, P. Mäder, H. Hu, A. Ghabi, L. Huang, J. Lü, and A. Egyed, "Can Method Data Dependencies Support the Assessment of Traceability Between Requirements and Source Code?" *J. Softw. Evol. Process*, vol. 27, no. 11, pp. 838–866, Nov. 2015. DOI: 10.1002/smr.1736

[42] H. Gao, H. Kuang, K. Sun, X. Ma, A. Egyed, P. Mäder, G. Rong, D. Shao, and H. Zhang, "Using Consensual Biterms from Text Structures of Requirements and Code to Improve IR-Based Traceability Recovery," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, Jan. 2023. DOI: 10.1145/3551349.3556948

[43] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically Enhanced Software Traceability Using Deep Learning Techniques," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 3–14. DOI: 10.1109/ICSE.2017.9

[44] W. Wang, N. Niu, H. Liu, and Z. Niu, "Enhancing Automated Requirements Traceability by Resolving Polysemy," in *2018 IEEE 26th International Requirements Engineering Conference (RE)*, Aug. 2018, pp. 40–51. DOI: 10.1109/RE.2018.00-53

[45] T. Zhao, Q. Cao, and Q. Sun, "An Improved Approach to Traceability Recovery Based on Word Embeddings," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, Dec. 2017, pp. 81–89. DOI: 10.1109/APSEC.2017.14

[46] C. Mills, J. Escobar-Avila, A. Bhattacharya, G. Kondyukov, S. Chakraborty, and S. Haiduc, "Tracing with Less Data: Active Learning for Classification-Based Traceability Link Recovery," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2019, pp. 103–113. DOI: 10.1109/ICSME.2019.00020

[47] M. Zhang, C. Tao, H. Guo, and Z. Huang, "Recovering Semantic Traceability between Requirements and Source Code Using Feature Representation Techniques," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, Dec. 2021, pp. 873–882. DOI: 10.1109/QRS54544.2021.00096

[48] M. Glinz, "On Non-Functional Requirements," in *15th IEEE International Requirements Engineering Conference (RE 2007)*, Oct. 2007, pp. 21–26. DOI: 10.1109/RE.2007.45

[49] F.-L. Li, J. Horkoff, J. Mylopoulos, R. S. S. Guizzardi, G. Guizzardi, A. Borgida, and L. Liu, "Non-functional requirements as qualities, with a spice of ontology," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, Aug. 2014, pp. 293–302. DOI: 10.1109/RE.2014.6912271

[50] J. Eckhardt, A. Vogelsang, and D. M. Fernández, "Are "non-functional" requirements really non-functional? an investigation of non-functional requirements in practice," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. Austin, Texas: Association for Computing Machinery, May 2016, pp. 832–842. DOI: 10.1145/2884781.2884788

[51] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, Jun. 1992. DOI: 10.1109/32.142871

[52] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Boston, MA: Springer US, 2000. DOI: 10.1007/978-1-4615-5269-7

[53] M. Broy, "Rethinking Nonfunctional Software Requirements," *Computer*, vol. 48, no. 5, pp. 96–99, May 2015. DOI: 10.1109/MC.2015.139

[54] M. Glinz, "A Glossary of Requirements Engineering Terminology," Jul. 2022.

[55] CoEST, "Center of Excellence for Software & Systems Traceability." [Online]. Available: http://coest.org/

[56] K. Krippendorff, *Content Analysis: An Introduction to Its Methodology*. SAGE Publications, May 2018.

[57] ——, "Reliability in Content Analysis," *Human Communication Research*, vol. 30, no. 3, pp. 411–433, 2004. DOI: 10.1111/j.1468-2958.2004.tb00738.x

[58] T. Hey, "Dataset for Requirements Classification in Traceability Link Recovery Datasets," Zenodo, Apr. 2023. DOI: 10.5281/zenodo.7867846