mathematics

*Article*

# A Cross-Level Requirement Trace Link Update Model Based on Bidirectional Encoder Representations from Transformers

**Jiahao Tian [1,2], Li Zhang [1,2] and Xiaoli Lian [1,2,*]**

1 School of Computer Science and Engineering, Beihang University, Beijing 100191, China
2 State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China
* Correspondence: lianxiaoli@buaa.edu.cn

**Abstract:** Cross-level requirement trace links (i.e., links between high-level requirements (HLRs) and low-level requirements (LLRs)) record the top-down decomposition process of requirements and support various development and management activities (e.g., requirement validation). Undoubtedly, updating trace links synchronously with requirement changes is critical for their constant availability. However, large-scale open-source software that is rapidly iterative and continually released has numerous requirements that are dynamic. These requirements render timely update of trace links challenging. To address these problems, in this study, a novel deep-learning-based method, deep requirement trace analyzer fusing heterogeneous features (DRAFT), was proposed for updating trace links between various levels of requirements. Considering both the semantic information of requirement text descriptions and the process features based on metadata, trace link data accumulated in the early stage are comprehensively used to train the trace link identification model. Particularly, first, we performed second-phase pre-training for the bidirectional encoder representations from transformers (BERT) language model based on the project document corpus to realize project-related knowledge transfer, which yields superior text embedding. Second, we designed 11 heuristic features based on the requirement metadata in the open-source system. Based on these features and semantic similarity between HLRs and LLRs, we designed a cross-level requirement tracing model for new requirements. The superiority of DRAFT was verified based on the requirement datasets of eight open-source projects. The average F1 and F2 scores of DRAFT were 69.3% and 76.9%, respectively, which were 16.5% and 22.3% higher than baselines. An ablation experiment proved the positive role of two key steps in trace link construction.

**Keywords:** cross-level requirement traceability; traceability maintenance; multi-features fusion; bidirectional encoder representations from transformers

**MSC:** 68N01; 46-04

check for updates

**Citation:** Tian, J.; Zhang, L.; Lian, X. A Cross-Level Requirement Trace Link Update Model Based on Bidirectional Encoder Representations from Transformers. *Mathematics* **2023**, *11*, 623. https://doi.org/10.3390/math11030623

Academic Editors: Rui Peng and Kaiye Gao

Received: 29 December 2022
Revised: 20 January 2023
Accepted: 22 January 2023
Published: 26 January 2023

## 1. Introduction

In complex software systems, requirements are decomposed layer by layer from top to bottom [1]. In this process, ensuring that each requirement at the high abstraction level is refined into a requirement at a lower level is critical. Each low-level requirement (LLR) should trace up to a specific high-level requirement (HLR); otherwise, subsequent design and implementation cannot satisfy system objectives or may exceed the system scope (over-standard) [2]. Many standards and norms such as DO-178C [3], IEEE Std. 830 [4], and CMMI [5] have emphasized the importance of requirement traceability to software development. In particular, DO-178C clearly stipulates the necessity to ensure that LLRs can satisfy HLRs and that each HLR is developed into LLRs subsequently. In an open-source system, developers cooperate across regions, and personnel mobility is strong [6]. Creating cross-level requirement trace links helps participants to swiftly understand the origin and development of requirements. Therefore, creating trace links between the requirements of different levels to support activities, such as requirement verification, validation, and change management, is crucial for ensuring that system development is correct.

With the constant gathering of requirements during system evolution, the requirement set continues to expand, and the integrity of trace links created in the early stage reduces. Untimely update of trace links mitigates their support for other activities and causes more errors [7]. However, manually updating trace links requires considerable manpower and material resources, and this phenomenon is especially obvious in open-source systems. Linus Torvalds (https://en.wikipedia.org/wiki/Linus_Torvalds, accessed on 25 January 2023), the father of Linux, proposed the principle of "Release early. Release often. And listen to your customers" for open-source software. This principle has the characteristics of short-cycle iterative development, rapid release, and continuous gathering of user requirements. In the constant iteration and release process, following the initial version release of a software project, new requirements from various origins such as new features, user feedback, and technical updates are frequently raised, and the number of trace links also increases rapidly [8]. In an open-source system featuring a short cycle, fast iterations, and high-frequency addition of new requirements, the cost of updating trace links during the evolution process is extremely high and may even exceed the cost of creating trace links at the initial stage of the project [9].

Although both academia and industry have recognized the importance of automated update and maintenance of trace links [10,11], few related studies have focused on it. Mader et al. [12] proposed to maintain the trace links between UML artifacts of different development activities (e.g., requirements and analysis) by capturing the relevant change events. Furthermore, existing studies on trace link maintenance focus on updating the trace links between requirements and code [9,13] and between requirements and Unified Modeling Language (UML) models [12]. However, limited studies have focused on different levels of requirements.

Most traceability-related studies have focused on automatic identification of requirement trace links. Existing methods typically use the text semantic analysis of the required artifacts to automatically create trace links. Mainstream methods include information retrieval-based methods (e.g., vector space model (VSM) [14], latent semantic indexing (LSI) [15], VSM-Part-of-Speech (POS) [16], VSM-Thesaurus [17], and Relevance feedback (RF) [18,19]), machine learning-based methods (e.g., methods mentioned in Refs. [20–23]), and deep-learning-based methods [24–26] (e.g., TraceNN [25] and TraceBERT [26]). Although trace link creation technologies can support the updating of trace links, only the textual distance between requirements is considered, and in many cases, the process-related information (e.g., writers and assigners) of requirement creation is ignored. Furthermore, the trace links of historical requirements are not comprehensively utilized. These two types of information are crucial for automatically creating trace links for new requirements.

As displayed in Figure 1, although the text descriptions (i.e., description and summary) of the high-level requirement JBIDE-26652 (https://issues.redhat.com/browse/JBIDE-26652, accessed on 25 January 2023) and the requirement JBIDE-26790 (https://issues.redhat.com/browse/JBIDE-26790, accessed on 25 January 2023) exhibit a low similarity, the two requirements highly overlap in terms of process data (e.g., assignee, creator, and components). A decomposition relationship exists between these two requirements, that is, cross-level traceability because the same author is very likely to decompose HLRs into LLRs after creating HLRs.

Figure 2 shows that historical requirements also help to trace link identification. For the pair of JBIDE-27384 (https://issues.redhat.com/browse/JBIDE-27384, accessed on 25 January 2023) and JBIDE-27673 (https://issues.redhat.com/browse/JBIDE-27673, accessed on 25 January 2023), the similarity between them is insufficient in both textual and process information. However, analysis revealed that JBIDE-27672 (https://issues.redhat.com/browse/JBIDE-27672, accessed on 25 January 2023) is an LLR traced to JBIDE-27384, which has a high similarity with JBIDE-27673 in terms of textual and process information. Therefore, a trace link is very likely to exist between JBIDE-27673 and JBIDE-27384.

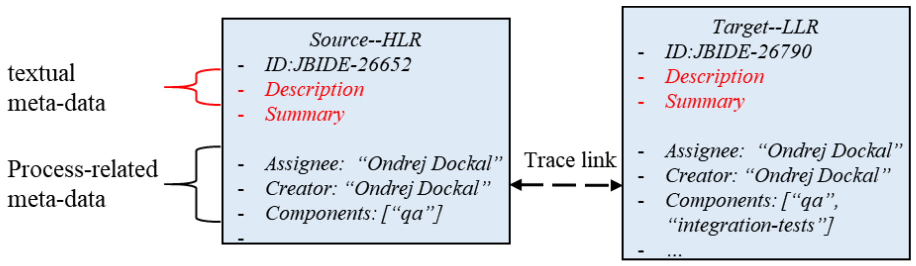

textual meta-data
Process-related meta-data
Source--HLR
- ID:JBIDE-26652
- Description
- Summary
- Assignee: "Ondrej Dockal"
- Creator: "Ondrej Dockal"
- Components: ["qa"]
Trace link
Target--LLR
- ID:JBIDE-26790
- Description
- Summary
- Assignee: "Ondrej Dockal"
- Creator: "Ondrej Dockal"
- Components: ["qa", "integration-tests"]
- ...

**Figure 1.** Example: indication effect of requirement process data for the creation of a trace link.

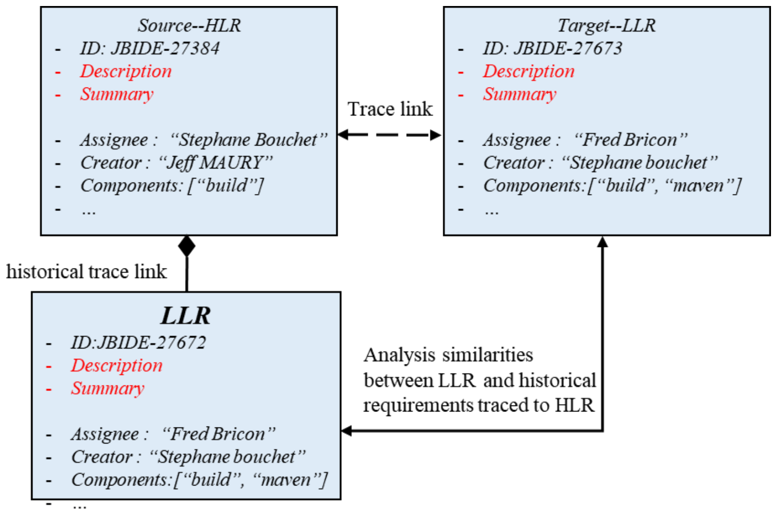

Source--HLR
- ID: JBIDE-27384
- Description
- Summary
- Assignee : "Stephane Bouchet"
- Creator : "Jeff MAURY"
- Components: ["build"]
- ...
Trace link
Target--LLR
- ID: JBIDE-27673
- Description
- Summary
- Assignee : "Fred Bricon"
- Creator : "Stephane bouchet"
- Components:["build", "maven"]
- ...
historical trace link
LLR
- ID:JBIDE-27672
- Description
- Summary
- Assignee : "Fred Bricon"
- Creator : "Stephane bouchet"
- Components:["build", "maven"]
- ...
Analysis similarities between LLR and historical requirements traced to HLR

**Figure 2.** Example: historical requirements help to identify new trace links.

Inspired by these two points, this study proposed a novel deep-learning-based method, deep requirement trace analyzer fusing heterogeneous features (DRAFT), for automatically updating trace links. This method can learn the trace the link identification model from historical data, automatically recommend candidate trace links for analysts for new requirements, and assist analysts in updating cross-level requirement trace links during requirement evolution. In DRAFT, the joint feature representation (i.e., text features and process features) of requirements is established from the perspectives of natural language description and process information. Based on the BERT model [27], DRAFT also integrates the direct feature extracted from the pairs of candidate requirements, as well as the extended features by retrieving the historical trace links to automatically develop trace links between cross-level requirements. In terms of capturing text semantic similarity, considering the semantic differences of terms in various contexts, we proposed to perform second-phase pre-training for the BERT language model to ensure the encoding of the required text is highly suitable for the project context. In terms of extracting process features, DRAFT introduces 11 heuristic features based on metadata and utilizes historical trace link data when extracting features. We collect requirement trace links from eight open-source projects to construct datasets and conduct experimental evaluations for DRAFT. The evaluation results revealed that DRAFT outperformed the existing baseline methods in identifying trace links.

The contributions of this paper are as follows:

1. A pre-trained model-based approach DRAFT is proposed for updating cross-level requirement trace links. Compared with existing studies, we extended the features into two dimensions. In terms of feature types, process features are added in addition to text features. In terms of requirement types, instead of directly analyzing the candidate requirement pairs (i.e., direct features), the requirements related to candidate requirements (i.e., extended features) are also analyzed.

2. An experimental evaluation is performed for eight open-source projects in various domains and scales. The results revealed that the performance of DRAFT is considerably better than that of the baseline methods such as VSM, relevance feedback, and TraceBERT. DRAFT achieved average F1 and F2 scores of 69.3% and 76.9%, which are up to 16.5% and 22.3% higher than those of the baselines, respectively.
3. The datasets and DRAFT-related code are made available online (https://gitee.com/ttstr/DRAFT, accessed on 25 January 2023).

The remainder of this paper is organized as follows: Section 2 describes the research background and defines the problems of requirement trace links update; Section 3 summarizes the relevant research status in detail; Section 4 introduces the overall framework of DRAFT; Sections 5–7 introduce the three core steps of DRAFT, i.e., project-specific pre-training, heuristic feature extraction, and the deep neural network architecture of the trace link identification model; Section 8 details an experimental evaluation and comparative analysis of the proposed method and the baseline methods; Section 9 discusses the validity threats and limitations of this study. Finally, a conclusion is given.

## 2. Research Background and Problem Definition

### *2.1. Research Background: Cross-Level Requirements Traceability*

To adapt to the loosely coupled and cross-regional cooperative development pattern, a lightweight, informal just-in-time requirement engineering [28] is adopted in open-source systems, and an issue log management system is used to record and manage requirements. Requirement development in an open-source system is a top-down decomposition process. At the beginning of a project, analysts define the HLRs that describe the long-term goals, with requirements known as "epics", of the project and decompose these into requirements such as "features" (or "feature requests"), "enhancements" (or "improvements"), and "tasks". Tasks are broken down into finer-grained sub-tasks. Some issue log types are predefined in the issue tracking system to record requirements [29]. The issue log types of requirements on most widely used JIRA (Issue management tools—popularity ranking (2017). https://project-management.zone/ranking/category/issue, accessed on 25 January 2023) are displayed in Figure 3. Requirements have three abstraction levels (from high to low): parent (including epics), standard (including features, enhancements, and tasks), and child (including sub-tasks).

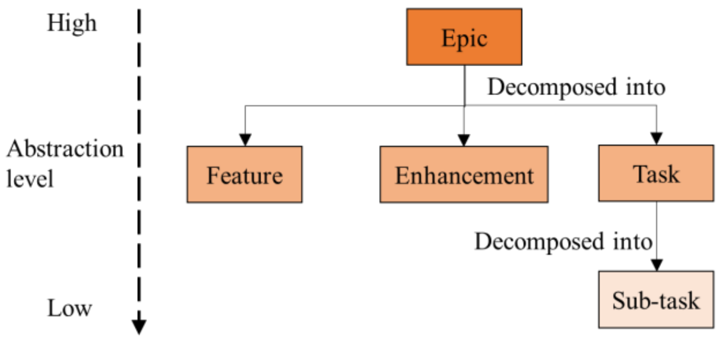


**Figure 3.** Requirements of different abstraction levels in JIRA and their decomposition relationships.

Cross-level requirement traceability means trace links between requirements in different abstraction levels. Given that traceability is primarily used to record the decomposition relationship between requirements, we focus on the trace links between the requirements of adjacent levels, i.e., parent–standard and standard–child in JIRA.

In an open-source system, the raw data of requirement issue logs contain rich textual information and process information [30]. The textual information of the issue log is contained in two fields: *summary* (a concise summary of the issue) and *description* (a detailed description of the issue) [31]. When manually constructing a requirement trace link, an analyst should read and understand the text descriptions such as *summary* and *description* of the requirement. Next, the analyst analyzes the semantic association between artifacts, which is the most intuitive basis for identifying trace links between requirements. Therefore, requirement traceability is a sentence-pair classification problem in natural language understanding, and the semantic similarity of a pair of requirements based on various technologies, such as information retrieval and deep learning language models, are measured to determine whether a trace link exists between artifacts.

Additionally, in the metadata of issue log, important process information, for example, people-related information fields, such as *creator* and *assignee* as well as *creation time*, *components*, and labels, is recorded. This type of information is a potential basis for analyzing requirement trace links. For example, requirements raised by the same developer/user are likely to correspond to the same or related functions. Thus, trace links are more likely to exist between requirements raised by the same developer/user.

Therefore, text semantic similarity is used for integrating the analysis of process features between cross-level requirements, which renders accurate identification of complete cross-level requirement trace links.

### 2.2. Problem Definition: Update of Requirement Trace Links

Because of feedback from end users, project technology update, and constant proposal of new requirements during the version evolution of software systems, analysts are required to update the trace links between requirements to ensure their continuous availability.

Figure 4 describes a trace link update task. Let a historical project requirement set $R_{hist} = \{HLRS_{hist}, \mathrm{LLRS}_{\mathrm{hist}}\}$, where $HLRS_{hist}$ is the HLR set and $LLRS_{hist}$ is the LLR set. $TLS_{hist}$ is a set of relationships between $HLRS_{hist}$ and $LLRS_{hist}$. Each element in $TLS_{hist}$ is a two-tuple $(hlr', llr')$, where $hlr' \in HLRS_{hist}$ is the source requirement (HLR entry), and $llr' \in LLRS_{hist}$ is the target requirement (LLR entry). When new requirements $R_{new}$ ($R_{new} = HLRS_{new} \cup LLRS_{new}$) are obtained, the existing trace link set $TLS_{hist}$ needs to be updated. This task is to construct the trace links between requirements of different level in $R_{new}$ and $R_{hist}$.

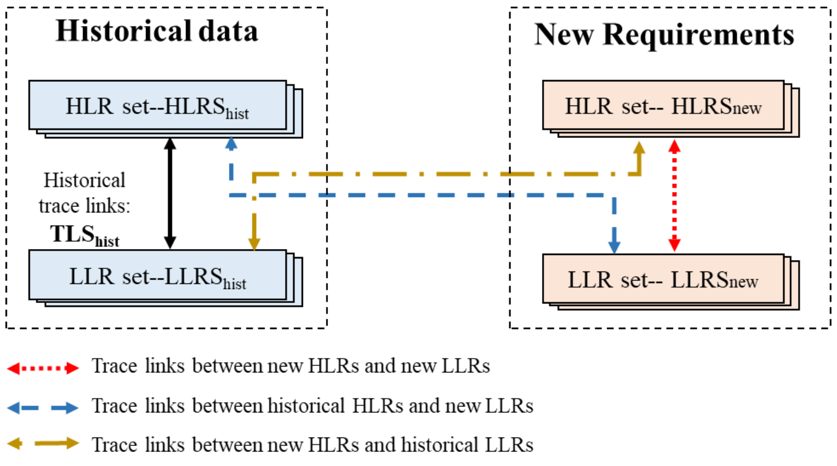

Historical data
HLR set--HLRS$_{hist}$
Historical trace links: **TLS$_{hist}$**
LLR set--LLRS$_{hist}$
New Requirements
HLR set-- HLRSnew
LLR set-- LLRSnew
Trace links between new HLRs and new LLRs
Trace links between historical HLRs and new LLRs
Trace links between new HLRs and historical LLRs

**Figure 4.** Identifying the source and target of the trace link for new requirements.

The following three cases may exist when a trace link is created for new requirements: between new HLR and historical LLR, between historical HLR and new LLR, and between new HLR and new LLR. The trace links defined by the aforementioned three tasks are $Trace_{<HLRnew,\ LLRhist>}$, $Trace_{<HLRhist,\ LLRnew>}$, and $Trace_{<HLRnew,\ LLRnew>}$. The set of trace links created for new requirements to be solved in this task, that is, $Trace_{new}$, is the union of the three sets. Thus, $Trace_{new} = Trace_{<HLRSnew,\ LLRShist>} \cup Trace_{<HLRShist,\ LLRSnew>} \cup Trace_{<HLRSnew,\ LLRSnew>}$.

Note that our proposed method can support the addition, deletion, and modification scenarios regarding the requirement evolution and is not limited to establishing a trace link for the newly added requirement. For requirements to be deleted, all related trace links should be deleted, which is not difficult technically. The modification of requirements is equal to the deletion of old requirements and the addition of new requirements (requirements after modification).

## 3. Related Work

The Trace Link Evolver proposed by Rahimi and Cleland-Huang in 2019 [9] can automatically update the trace link between requirements and code during the system iteration process. They first analyzed 24 common scenarios of code change and defined the trace link evolution rules for each change scenario to update the trace link. In 2012, Mader [12] et al. proposed a semi-automatic approach for maintaining the trace links between requirements and design models expressed in UML, which can update the trace links with the progress of development activities. Under this framework, a specific UML modeling tool was used to capture the flow of change events caused by various development activities, and heuristic rules are predefined for development activities, leading to automatic updating of trace links. In addition, few studies have been conducted on trace link update for cross-levels of requirements. Most studies have focused on the updating of trace links between requirements and codes and between requirements and UML models.

However, in the requirement tracing domain, numerous studies have focused on automatically identifying and creating trace links. Such methods can ensure the continuous availability of trace links by regularly recreating trace links during project version iterations. Rath et al. [23] proposed a machine-learning-based method for identifying trace links between requirement problems and submission records of open-source systems. This method not only calculates the text similarity between artifacts based on the VSM [14] model but also considers process-related attributes such as stakeholder information and timing relationships. Weka's J48 decision tree [32] training model was applied to verify the identification effect when using various feature sets. Their experimental results revealed that the best results can be achieved when both similarity and process features are used. We incorporated their ideas and analyzed the heuristic features related to the process while considering the text description content of requirement entries as the main identification basis.

The semantic similarity of the text description is the most intuitive basis for creating trace links between requirements. Most studies have only relied on the text descriptions of requirements to identify trace links. The information retrieval technique and learning-based methods have been adopted for identifying the requirement trace links based on the text features of requirements. Early methods for automated requirement trace include classic VSM [14] and LSI [15] that determine the trace link by capturing the same words used in the text descriptions of source and target artifacts and calculating the similarity between the two text vectors. However, in practice, artifacts are written by various people or organizations, and different words may have been used to describe the same concept. Thus, a term mismatch problem may occur, which is the primary concern of this type of method [33]. To address this problem, adding semantic information improved identification. For example, thesaurus [17] and domain knowledge [34] can help to capture the semantic association between different words based on the vocabulary support. The relevance feedback technique [19] can improve the query statement and expand the scope of semantic retrieval based on user feedback on information retrieval results, which increases accuracy. With the development of artificial intelligence, methods based on machine learning and deep learning have gradually received considerable research attention. Learning-based methods typically regard the identification of requirement trace links as a binary classification problem. In methods based on machine learning, first, semantic features are extracted from the requirement descriptions, and models such as Naive Bayes [35] and random forest [36] are used to predict the trace link between requirements pairs.

Methods based on deep learning can automatically embed text features through a deep neural network, which reduces the dependence on manual selection in the feature representation stage in case of machine learning methods. The recurrent neural network (RNN) is widely used in natural language processing. The method can embed contextual semantic information into word vectors. The RNN has a stronger ability to represent text than the information retrieval and machine learning methods [37]. Guo et al. proposed the TraceNN method [25] to trace requirements to design documents. In TraceNN, first, the features of text sequences are embedded based on the RNN, and multilayer perceptron (MLP) is then used to complete the classification of trace links. They evaluated two types of RNN models, namely long short-term memory (LSTM) and gated recurrent unit (GRU) on large-scale industrial datasets. Their results showed that GRU delivered better results in terms of mean average precision. However, in RNNs, usually only one side of the context information is encoded because of its unidirectional structure. With the increase in the sequence length, the embedded context information gradually weakens [37]. In 2018, Devlin et al. proposed BERT [27] based on the transformers [38], which solved this problem satisfactorily. This method achieved state-of-the-art results in a series of natural language processing tasks. BERT consists of two stages, namely pre-training and fine-tuning on downstream tasks. In the pre-training stage, unsupervised task training is performed on a large corpus. The semantic knowledge in the corpus is encoded into the language model, which is used to embed the text representation vector. This representation is applied to downstream tasks. Native BERT is trained on general corpora such as Wiki. Currently, numerous corpora have been developed for various domains and trained domain-specific language models such as BioBERT [39], FinBERT [40], and CodeBERT [41]. In the requirement trace field, TraceBERT [26], proposed by Lin et al. in 2021, investigated the application effect of BERT in tracing requirements and codes. This is the first study that applied BERT or other transformer-based methods to software traceability tasks. They performed second-phase pre-training on large datasets of similar tasks and subsequently transferred domain knowledge into the language model. The model was then fine-tuned and applied to the downstream task "issue (natural language)–commit (programming language)" to improve the trace effect. They evaluated three commonly used BERT architectures (i.e., *single*, *twin*, and *Siamese*) on open-source projects. Their experimental results showed that *single* architecture achieves the best accuracy, while *Siamese* architecture achieved similar accuracy with faster training time [26]. Considering that the requirement trace task is typically a project-specific task, we also developed a corpus from project-related documents based on transfer learning. After second-phase pre-training, we encoded the contextual semantic information and knowledge of the projects into the language model to improve the quality of text representations in the text embedding stage and subsequently achieved superior results in cross-level requirement trace activities.

Studies have proved that the deep learning-based language model BERT achieved excellent performance in downstream tasks of natural language understanding. Refs. [23] and [42] have confirmed that the introduction of process features can improve the quality of trace link identification. However, the two methods are yet to be combined in a study. This study combined these two methods. First, the adaptability of the BERT model to the project context was improved through the second-phase pre-training on the project corpus and used as an encoder for the required text. Second, the trace link in the historical version was fully utilized to extract heuristic features based on the requirement metadata. We constructed a deep neural network to fuse the two kinds of features and improved the effect of trace link identification.

## 4. DRAFT Framework

To comprehensively analyze the correlation between cross-level requirements, the features of requirements in terms of text description are analyzed, and information, such as components, task labels, and stakeholders, is processed based on the metadata of issue logs. A network architecture DRAFT that integrates text features and process features is then designed. Based on the BERT language model, DRAFT allows for the embedding and joint feature representation of heterogeneous features, which renders the creation of cross-level requirement trace links for new requirements.

As displayed in Figure 5, the DRAFT architecture includes three key components: the project-specific BERT second-phase pre-training module, the heterogeneous feature extraction module, and the trace link identification model that integrates heterogeneous features. These three components are executed in sequence. Trace links can be created for new requirements by training the trace link identification model.

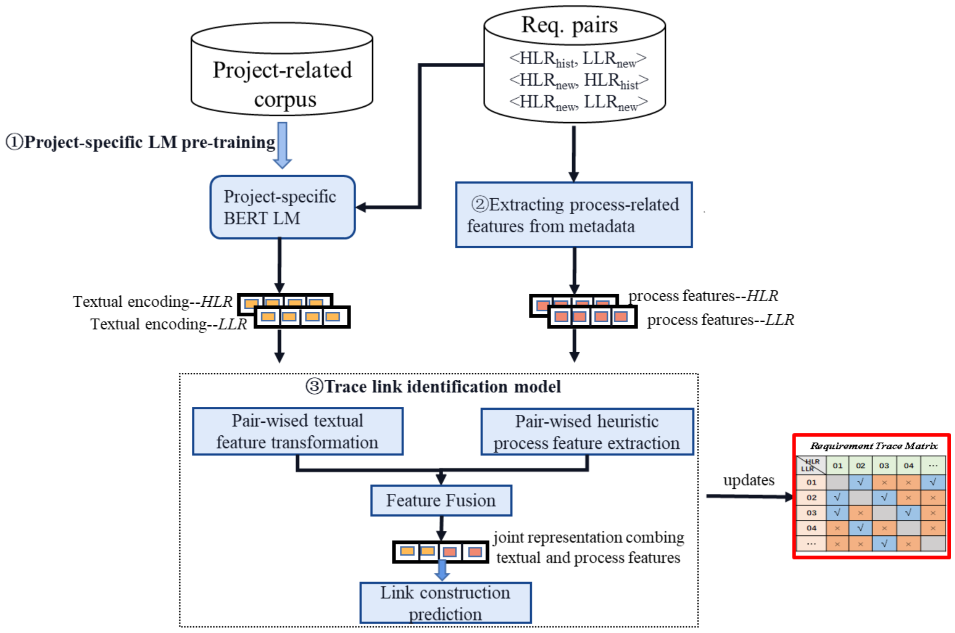

Project-related corpus
Req. pairs
<HLR_hist, LLR_new>
<HLR_new, HLR_hist>
<HLR_new, LLR_new>
①Project-specific LM pre-training
Project-specific BERT LM
②Extracting process-related features from metadata
Textual encoding--HLR
Textual encoding--LLR
process features--HLR
process features--LLR
③Trace link identification model
Pair-wised textual feature transformation
Pair-wised heuristic process feature extraction
Feature Fusion
joint representation combing textual and process features
Link construction prediction
updates
Requirement Trace Matrix

**Figure 5.** Flowchart of DRAFT.

In this study, the BERT pre-training model [27] was selected to embed requirement text and features. We selected BERT because its performance is excellent in various tasks of natural language understanding (such as question answering and sentence-pair classification) [37]. Furthermore, Lin et al. [24,26] used considerable data and proved that excellent transfer ability and context understanding ability of BERT make their approach TraceBERT more effective in establishing a trace link between commit and code than baseline methods such as the VSM and LSTM. However, the BERT pre-training model runs on general corpora. To enhance its adaptability to projects and domains and improve its ability to understand domain-specific corpora, we collected all text descriptions related to the requirements in the projects. A second-phase pre-training was performed for BERT to obtain a project-specific language model.

The second step is to identify and embed the heterogeneous features of the requirements. In most current requirement trace methods based on deep learning, only the text features of requirements are considered [17–22,24–26,30,34]. The proposed DRAFT method fully utilizes the limited historical trace data to obtain a complete representation of requirement features. The features of process information, such as the requirement creator and the creation time, were also considered when using the natural language descriptions of requirements as the basis for trace link identification. In addition to directly extracting the text features and process features between the requirements in pairs, we retrieved the historical trace list of each requirement from the historical trace link and analyzed the extended features between the requirements in the historical trace list and the newly added requirements.

Finally, a trace link identification model fusing heterogeneous features was constructed. The model consists of three modules, namely requirement-pair feature embedding layer, feature fusion layer, and trace link identification layer. The primary function of the requirement-pair feature embedding layer is to embed the text features and heuristic features of HLR-LLR pairs based on the aforementioned feature extraction method. Textual features are high-dimensional (200-dimensional), whereas the heuristic process features are multiple low-dimensional (1-dimensional) features. The feature fusion layer reduces the dimensionality of high-dimensional text features based on cosine similarity and concatenates them with low-dimensional features to realize the fusion of heterogeneous features. Finally, the fused features are input into the trace link identification layer to obtain the trace link identification result of the pair of requirements.

## 5. Project-Specific Pre-Training

In this study, we designed DRAFT upon BERT language model [27] for the following reasons. Firstly, the BERT architecture is based on transformers [38]. Compared to classic unidirectional models such as RNN, when embedding a word in a given sequence, BERT is capable of encoding the surrounding context bi-directionally. Secondly, the training process of BERT can be parallel. This allows BERT to obtain a more sufficient context vector with much less time consumption. Thirdly, BERT has been successfully applied to requirement traceability problems of open-source projects in recent studies [26,30] and outperformed two popular RNN baselines (i.e., GRU and LSTM).

The currently BERT pre-trained language models are trained on general corpora such as WordPiece and Wiki. Therefore, the context of the corpora differs from the technical documents of the project. Ref. [26] proved that a second phase of pre-training using a domain corpus (i.e., domain-adaptive pre-training) could lead to performance gains. Thus, to obtain a language representation model that can understand project-related documents more accurately and enhance the ability of the model to represent domain/project-related vocabulary, the natural language text contained in the *summary* and *description* fields of all requirements (including HLRs and LLRs) for each project was extracted to construct a project-related corpus. We performed second-phase pre-training for the BERT model "uncased_L-12_H-768_A-12" (http://github.com/google-research/bert, accessed on 25 January 2023) based on two unsupervised tasks: masked language model (MLM) task and next sentence prediction (NSP) task.

The MLM task randomly masks some words in the original text to construct a training set. The training goal is to allow the encoder to predict the masked words based on the context. The MLM prediction task enables the model encoding result to contain the context information. The following original MLM training strategy of BERT was adopted: (1) 15% of the tokens in the sentence are randomly selected; (2) among the 15% tokens, 80% are replaced by "[mask]", 10% remain unchanged, and the remaining 10% are replaced with a random token; (3) the tokens selected in the first step are predicted based on the context.

NSP is used to train the sentence-level feature extraction ability of BERT, which is a sentence-pair classification task. Thus, given a pair of input sentences S1 and S2, NSP predicts whether S2 is the next sentence of S1. The NSP task was selected to increase the ability of the language model to understand sentence relationships. For the NSP task, first, the paragraphs and sentences of the long natural language descriptions are identified in the corpus based on the Stanford CoreNLP tool. Next, sentence pairs with a sliding window of length 2 are extracted for each natural language description. Finally, the sentence pairs in the original order are considered positive samples, whereas sentence pairs in the reversed order are considered negative samples to construct a training set.

After second-phase pre-training, a project-specific pre-training model was obtained to extract the features of natural language descriptions in the requirements.

## 6. Heuristic Feature Extraction Based on Metadata

Based on the three scenarios defined in Section 2.2, the following three types of requirements pairs should be considered when updating trace links between different levels of requirements: (a) new HLR and historical LLR, $hlr_{new}$–$llr_{hist}$; (b) historical HLR and new LLR, $hlr_{hist}$–$llr_{new}$; (c) new HLR and new LLR, $hlr_{new}$–$llr_{new}$. This section designs 11 features of cross-level requirement pairs for these three scenarios, including process features and text feature (Table 1).

**Table 1.** Metadata-based feature extraction in DRAFT.

| ID | Feature Type | Feature Name | Applicable Requirement Pair | Explanation |
|---|---|---|---|---|
| Ft.1 | Process feature | same_coms | * (a), * (b), and * (c) | The normalized representation of the number of *components* shared by a pair of cross-level requirements. (See Section 6.1) |
| Ft.2 | | find_coms | | The maximal degree that the *component* labels of one requirement can be covered by the *summary* and *description* of the other one. (See Section 6.1) |
| Ft.3 | | same_labels | | The normalized representation of the number of *labels* shared by a pair of cross-level requirements. (See Section 6.2) |
| Ft.4 | | same_creator | | Whether the two requirements have the same *creator*, i.e., if *hlr.creator = llr.creator.* (See Section 6.2) |
| Ft.5 | | same_assignee | | Whether the two requirements are assigned by the same *assignee*, i.e., if *hlr.assignee = llr.assignee.* (See Section 6.2) |
| Ft.6 | | same_stk | | Whether the *creator* of one requirement is equal to the *assignee* of the other one, i.e., if *hlr.creator = llr.assignee or hlr.assignee = llr.creator.* (See Section 6.2) |
| Ft.7–10 | | extended_features | (a) and (b) | Extracting Ft. 1, Ft.2, Ft.4, Ft.5 between $*llr_{hist}.tracedReq$ and $hlr_{new}$ (in scenario a), or between $*hlr_{hist}.tracedReq$ and $llr_{new}$ (in scenario (b)). (See Section 6.3) |
| Ft.11 | Text feature | text_feature | (a), (b), and (c) | Text embedding of *hlr.description, hlr.summary, llr.description* and *llr.summary.* (See Section 6.4) |

* (a): a pair of new hlr and historical llr, $hlr_{new}–llr_{hist}$; * (b): a pair of historical hlr and new llr, $hlr_{hist}–llr_{new}$; * (c): a pair of new hlr and new llr, $hlr_{new}–llr_{new}$ $*llr_{hist}.tracedReq$: historical *hlrs* traced to $llr_{hist}$; * $hlr_{hist}.tracedReq$: historical *llrs* traced to $hlr_{hist}$.

In Table 1, Ft.1–Ft.10 are process features. For a given pair of cross-level requirements hlr and llr, Ft.1–Ft.6 are extracted directly from the metadata of this requirement pair by using the methods mentioned in Sections 6.1 and 6.2. Further, to fully utilize historical trace link, we designed extended features Ft.7–10 for two requirement pairs ($hlr_{new}–llr_{hist}$ and $hlr_{hist}–llr_{new}$) and extract process-related features (same_coms, find_coms, same_creator and same_assignee) between $hlr_{new}$ and $llr_{hist}.tracedReq$ (or $hlr_{hist}.tracedReq$ and $llr_{new}$). Here, $llr_{hist.}tracedReq$ and $hlr_{hist}.tracedReq$ are the lists of $hlr_{hist}$ and $llr_{hist}$ requirements that have trace links with $llr_{hist}$ and $hlr_{hist}$, respectively, according to the historical trace links $TLS_{hist}$. They can be defined as follows:

$$hlr_{hist}.tracedReq = \{llr_{hist} \mid hlr_{hist} \in HLRS_{hist} \cap llr_{hist} \in LLRS_{hist} \cap <hlr_{hist}, llr_{hist}> \in TLS_{hist}\};$$

$$llr_{hist}.tracedReq = \{hlr_{hist} \mid llr_{hist} \in LLRS_{hist} \cap hlr_{hist} \in HLRS_{hist} \cap <hlr_{hist}, llr_{hist}> \in TLS_{hist}\} \quad (1)$$

where $HLRS_{hist}$ and $LLRS_{hist}$ are the sets of HLRs and LLRs in the historical version, and $TLS_{hist}$ is a set of historical trace links.

Ft.11 is a set of text embedding features which we extracted from the natural language descriptions (typically included in two metadata fields, *summary* and *description*) of the requirement using the language model obtained in Section 5.

### *6.1. Heuristic Features Related to Components and Labels*

The component attribute of a requirement marks the specific components that the requirement targets. If two cross-level requirements share the same component, a trace link is more likely to exist between them. This likeliness is also true for a pair of cross-level requirements with the same labels. Therefore, for a pair of cross-level requirements (hlr, llr), heuristic features based on the *components* and *labels* fields of the requirements should be designed. Based on these considerations, the features (same_coms and find_coms) based on the *components* field and the feature (same_labels) based on the *labels* field are extracted.

The feature same_coms is a floating-point number in the range of *0~1.0* and is a normalized representation of the number of components shared by a pair of cross-level requirements, as presented in Equation (1). The numerator is the number of components that appear in the component lists of both HLRs and LLRs, and the denominator is the maximum length of the component lists of HLRs and LLRs. To prevent the phenomenon of dividing by 0 when the component lists of both HLRs and LLRs are empty, we add 1 to the denominator.

$$pair.same_{coms} = \frac{|hlr.coms \cap\ llr.coms|}{\max(|hlr.coms|,\ |llr.coms|) + 1} \tag{2}$$

We extract the find_coms feature, which reflects how many keywords in the component list of one requirement can be obtained in the summary of the other requirement in a pair of cross-level requirements. The find_coms feature of a requirement pair is obtained by considering the maximum value from both find_hcoms and find_lcoms, as presented in Equation (2). In this equation, find_hcoms is the normalized representation of the number of keywords in the component list of the HLR appearing in the LLR summary, as shown in Equation (3). Here, ∣llr.sum+llr.des∣ indicates the number of words in the summary and description of llr. Similarly, Equation (3) can be used to calculate find_lcoms, that is, the number of keywords in llr.components appearing in hlr.summary and hlr.description, and then find_coms can be obtained.

Similarly, the feature same_labels is designed for labels. Its extraction method is the same as the extraction methods of same_coms.

$$pair.find_{coms} = \max(find_{hcoms},\ find_{lcoms}) \tag{3}$$

$$pair.find_{hcoms} = \frac{|\{com|com \in hlr.coms\ and\ com\ in\ (llr.\ sum + llr.des)\}|}{|llr.sum + llr.des| *\ (|llr.coms| + 1)} \tag{4}$$

### *6.2. Heuristic Features of Stakeholder Information*

The person making the request could be the project developer or the end user. A pair of cross-level requirements with the same stakeholder indicates that their themes or contents are correlated to some extent, and they are more likely to have a trace link.

To mark the correlation between a pair of cross-level requirements in terms of stakeholders, we extract the following three Boolean features to record whether the pair of requirements are raised by the same user/developer (same_creator), whether they are assigned to the same person to handle (same_assignee), and whether the stakeholder (same_stk) is the same.

$$pair.same\_creator = (hlr.creator == llr.creator)?1 : 0 \tag{5}$$

$$pair.same\_assignee = (hlr.assignee == llr.assignee)?1 : 0 \tag{6}$$

$$pair.same_{stk} = (hlr.creator == llr.assignee\ OR\ hlr.assignee == llr.creator)?1 : 0 \tag{7}$$

### *6.3. Extended Features Based on Historical Trace Links*

Because of the intricate relationship between requirements in the same system, some association may exist between new requirements and existing requirements. This association is helpful for creating traces for new requirements. Therefore, extended features were designed based on the historical trace list. As displayed in Figure 6, when determining whether a trace link exists between the new $LLR_{new}$ and the existing $HLR_{hist}$, the features between these two requirements (i.e., direct features) should be extracted. If $LLR_{new}$ has high similarity to one or more LLRs related to the $HLR_{hist}$, the probability that $LLR_{new}$ has a trace link with $HLR_{hist}$ is also high. Therefore, we use LLRs contained in $HLR_{hist}.traced\_reqs$ (i.e., the historical trace list of $HLR_{hist}$) with $LLR_{new}$ to form requirement pairs and extract features such as same_coms, find_com, same_creator, and same assignee by pairs as extended features (Ft.7–10). The vectors of direct features and extended features were concatenated into the final feature vector between $LLR_{new}$ and $HLR_{hist}$.

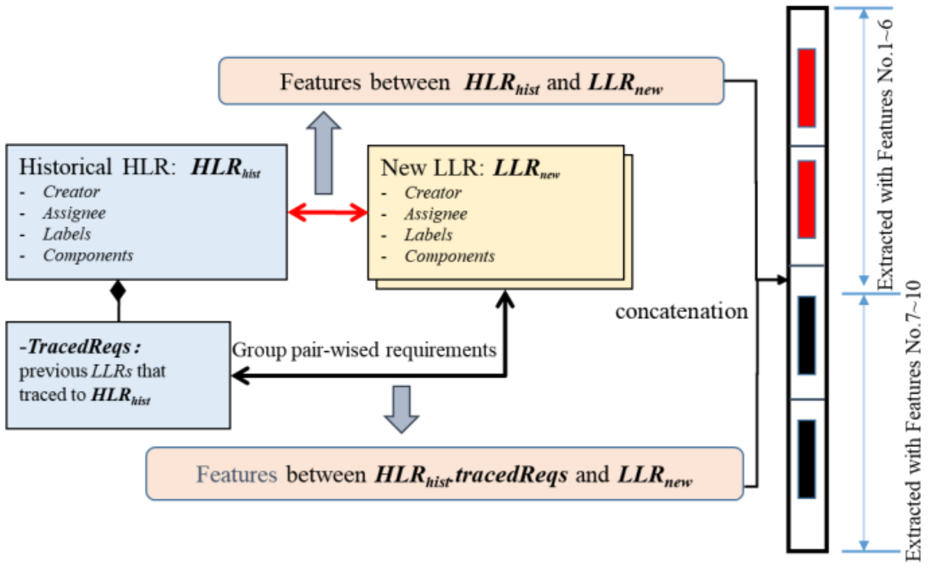

Features between HLR_hist and LLR_new
Historical HLR: HLR_hist
- Creator
- Assignee
- Labels
- Components
New LLR: LLR_new
- Creator
- Assignee
- Labels
- Components
-TracedReqs: previous LLRs that traced to HLR_hist
Group pair-wised requirements
concatenation
Features between HLR_hist.tracedReqs and LLR_new
Extracted with Features No.1~6
Extracted with Features No.7~10

**Figure 6.** Extract extended features based on the historical trace list.

Similarly, when identifying a trace link between a pair of requirements $hlr \in HLRS_{new}$ and $llr \in LLRS_{hist}$, extended features based on the historical trace list of llr can be extracted.

### *6.4. Textual Feature*

The natural language descriptions contain primary semantic information of requirements, and textual feature is an important factor when identifying trace links. Textual contents of a requirement mainly lie in the metadata attributes *summary* and *description*. Therefore, for a pair of *hlr* and *llr*, we extract the text features (Ft. 11) by embedding texts in *hlr.summary, hlr.description, llr.summary* and *llr.description*, utilizing the second-phase pre-trained project-specific language representation model $BERT_{pjt}$ (seen in Section 5).

## **7. Trace Link Identification Model Fusing Heterogeneous Features**

To support the joint analysis of high-dimensional text features of requirement description and heuristic features based on metadata, a neural network model that fuses heterogeneous features was designed.

### *7.1. Model Structure*

As displayed in Figure 7, the model typically includes three layers, namely requirement-pair feature embedding, heterogeneous feature fusion, and trace link identification layers.

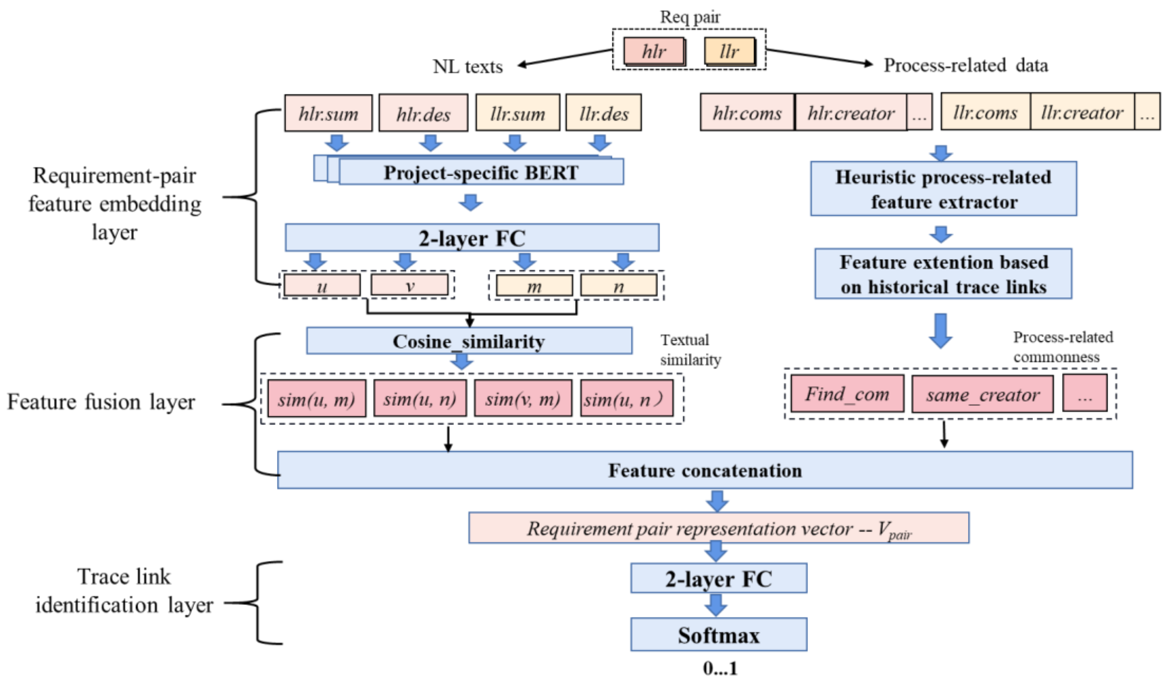


**Figure 7.** Neural network structure of DRAFT, which consists of three modules, that is, requirement-pair feature embedding, feature fusion, and trace link identification layers.

1. **Requirement-pair feature embedding layer.** This layer embeds text features and process features for the input cross-level requirement pair <hlr, llr> using the feature extraction method described in Section 6.

First, the text features of requirements are embedded by the second-phase pre-trained project-specific language representation model $BERT_{pjt}$ (seen in Section 5). For a serialized text $T = \{t1,t2 \ldots\ tn\}$, the last layer of $BERT_{pjt}$ (embedding layer) can output the word embedding $\{E_{CLS},E_{t1},E_{t2}..E_{tn},\ E_{SEP}\}$ of each word in $T$, where $E_{\mathrm{i}}$ is a 768-dimensional vector. This embedding is typically used as a representation of sentences for downstream tasks. Nils Reimers [43] et al. argued that three most commonly used strategies are as follows: (1) use $E_{\mathrm{CLS}}$ directly; (2) use the average pooling strategy, that is, calculate the average value of the representation vector corresponding to each word in $T$ to obtain $E_{\mathrm{mean}}$; (3) use the maximum pooling strategy, that is, calculate the maximum value of all word vectors in $T$ in each dimension to obtain $E_{\mathrm{max}}$. This study revealed that the strategy of using $E_{\mathrm{mean}}$ as the input of the downstream sentence-pair relationship classification task yields the best performance. Therefore, we adopted this strategy, as presented in Equation (7). To improve the adaptability of the sentence representation output by the pre-training model to the described requirement trace task, we added a fully connected layer after the pooling layer. After $E_{\mathrm{mean}}$ passes through the fully connected layer, the embedding vector $E_{\mathrm{T}}$ can be obtained for downstream tasks, as presented in Equation (8). In this equation, $W$ is a $k \times j$-dimensional trainable parameter, $j$ is the dimension of $E_{\mathrm{mean}}$, and $k$ is the dimension of the output text representation vector, which is set to 200.

$$E_{mean} = mean_{pooling(BERT_{pjt}.encoder(T))} \tag{8}$$

$$E_T = W * E_{mean} \tag{9}$$

The text content of a requirement is stored in the summary and description fields. As displayed in Figure 7, after the natural language descriptions of HLRs and LLRs, that is, *hlr.sum*, *hlr.des*, *llr.sum*, and *llr.des*, pass by the feature embedding layer, we obtain four 200-dimensional sentence representation vectors, namely $\vec{u}$, $\vec{v}$, $\vec{m}$, and $\vec{n}$, and use these as abstract text features of sentences.

Second, based on keywords and process information, we extract the common features of low-dimensional heuristic requirement pairs such as find_coms and same_creator using the method mentioned in Section 6. If a historical trace link between the HLR and LLR in a requirement pair exists, then extended features should be extracted based on the historical trace list. Finally, 10 one-dimensional heuristic features $F_{heu}$ were obtained.

2. **Heterogeneous feature fusion layer.** The feature fusion layer is used to fuse text features and heuristic features with various dimensions to comprehensively analyze the commonality of a pair of requirements in terms of text semantics and process data. After the processing at the feature embedding layer, the 200-dimensional text embedding representation vector is obtained in the natural language description, whereas heuristic features are one-dimensional.

The text embedding vector is an abstract feature obtained through a deep network. Heuristic features are shallow features, which can directly represent the common features of the HLR and LLR. To fuse the two types of features, the cosine similarity layer is used to reduce the dimensionality of the text vector. In particular, the cosine distance between the summary and description representations $\vec{u}$ and $\vec{v}$ of the HLR and the commendation vectors ($\vec{m}$ and $\vec{n}$) of the summary and description fields of the LLR is calculated to obtain four similarity values: $sim(\vec{u}, \vec{m})$, $sim(\vec{u},\vec{n})$, $sim(\vec{v},\vec{m})$, and $sim(\vec{v},\vec{n})$. The similarity value represents the semantic similarity between the requirements in the requirement pair <hlr, llr>, and $F_{heu}$ represents the commonality of the pair of requirements in terms of process features, as presented in Equation (9). Next, the similarity value was fused with the heuristic low-dimensional feature $F_h$ at the concatenation layer to obtain a complete requirement-pair feature representation $F_{pair}$, with its dimension being 14 (4 similarities plus 10 heuristic features).

$$F_{pair} = concat\left(\left[\mathrm{sim}\left(\vec{u},\vec{m}\right),\mathrm{sim}\left(\vec{u},\vec{n}\right),\mathrm{sim}\left(\vec{v},\vec{m}\right),\mathrm{sim}\left(\vec{v},\vec{n}\right)\right],F_{heu}\right) \tag{10}$$

3. **Trace link identification layer.** This layer includes two fully connected layers and one Softmax output layer. As presented in Equation (10), the feature $F_{pair}$ of the requirement pair in the previous step is used as the input, and the trace link identification result $C_{pair}$ of this pair of cross-level requirements is output. Here, $C_{pair}$ is one-dimensional, and takes the value of 0 (no trace link) or 1 (with a trace link), and W is a 1 × 10-dimensional trainable parameter.

$$C_{pair} = softmax(\ \sigma(\ W * F_{pair})\ ) \tag{11}$$

### *7.2. Loss Function*

We determine whether a trace link exists between a pair of requirements based on the joint feature representation of the pair of requirements in text information and process information and consider the identification of cross-level trace link to be a binary classification problem. Therefore, cross entropy (CE) was selected as the loss function to train the trace link identification model. The training goal is to minimize this loss. CE measures the difference between the true distribution and the predicted distribution of a random variable, as defined in Equation (11). In this equation, $m$ is the number of samples; $n$ is the number of class labels of the samples; $y_{ij}$ is the true probability that the class of sample $i$ is label $j$; and $y'_{ij}$ is the probability of the neural network predicting the class of sample $i$ to be label $j$.

$$CE = -\sum_{i=1}^{m}\sum_{j}^{n} y_{ij}\ln\left(y'_{ij}\right) \tag{12}$$

In the presented binary classification problem scenario, the real label of the sample can only take 1 (with a trace link) or 0 (no trace link) values. The binary cross entropy (BCE) can be defined as follows:

$$BCE_{loss} = -\sum_{i=1}^{m}\left(y_i * \ln\left(y'_i\right) + (1 - y_i) * \ln\left(y'_i\right)\right) \tag{13}$$

In requirement trace, the number of positive and negative samples is highly unbalanced. Negative samples far exceed positive samples. This phenomenon is severe in projects with more requirements. The ratio of negative samples to positive samples in the eight projects selected in this study is approximately 300:1 on average. This ratio is the highest in the JBIDE project, reaching 570:1. In this case, predicting the sample as the majority class (negative sample here) causes a smaller loss. Thus, the model always tends to predict a pair of cross-level requirements as having no trace link, which causes the model to fail. To solve this problem, a weight coefficient was set for the BCE loss function according to the ratio of positive and negative samples, and the loss weight of positive examples is increased as follows:

$$BCE_{weighted_{loss}} = -\sum_{i=1}^{m}\left(y_i * \ln\left(y'_i\right) * \alpha + (1 - y_i) * \ln\left(y'_i\right) * \beta\right) \tag{14}$$

where $\alpha$ and $\beta$ are calculated using the ratio of positive and negative samples, respectively. If the numbers of positive and negative samples are $n_{\text{pos}}$ and $n_{\text{neg}}$, respectively, then $\alpha = (n_{\text{pos}} + n_{\text{neg}})/n_{\text{pos}}$, and $\beta = (n_{\text{pos}} + n_{\text{neg}})/n_{\text{neg}}$.

## 8. Experimental Evaluation

Experimental evaluation was performed to verify the effectiveness of our DRAFT. We selected eight open-source projects involving different domains and collected requirements and trace links from their issue tracking systems. The evaluation was conducted from two aspects. First, the overall effects of DRAFT and the baseline methods in cross-level trace link identification were compared, and the strengths and weaknesses of the proposed method were analyzed. Then, two ablation experiments were performed to study the effects of two key designs of pre-training and extended features.

### *8.1. Objectives of Experimental Evaluation*

**RQ1** (overall evaluation): How is the performance of DRAFT in identifying cross-level trace links for new requirements? Is DRAFT better than the baseline methods?

**RQ2** (ablation experiment): Do the project-specific second-phase pre-training in DRAFT and the heuristic feature extraction based on metadata play a positive role?

### *8.2. Data Acquisition*

This sub-section details the project selection and the process of collecting raw data (HLRs, LLRs, and trace links).

**Project selection.** Following the process in Figure 8, we selected eight open-source projects under the Apache (https://issues.apache.org/jira/, accessed on 25 January 2023) and Redhat (https://issues.redhat.com/, accessed on 25 January 2023) and constructed a dataset for each project by collecting requirements and trace links from their issue tracker. First, based on the API provided by JIRA, we obtained all projects under Apache and Redhat that use JIRA as the issue tracker and have been active for more than three years and collected their issue logs and trace link data between issue logs. Next, we counted the number of cross-level trace links in each project and selected the projects with a link size of more than 600. Because this study investigated the trace link prediction method for new requirements in the continuous release process of a project, the verification data should have clear version information to distinguish historical requirements from new requirements. Therefore, we selected the projects in which the version number and release time were clearly recorded. To improve the generalizability of the experimental conclusions, we manually screened projects from various domains and finally obtained the following eight projects: Apache's Beam, CB (apache cordova), and Redhat's FH (feedHenry), JBIDE (jbosstools), AAH (automation hub), KEYLOACK, KOGITO, and PROJQUAY (project quay). Details of these eight projects are presented in Table 2.

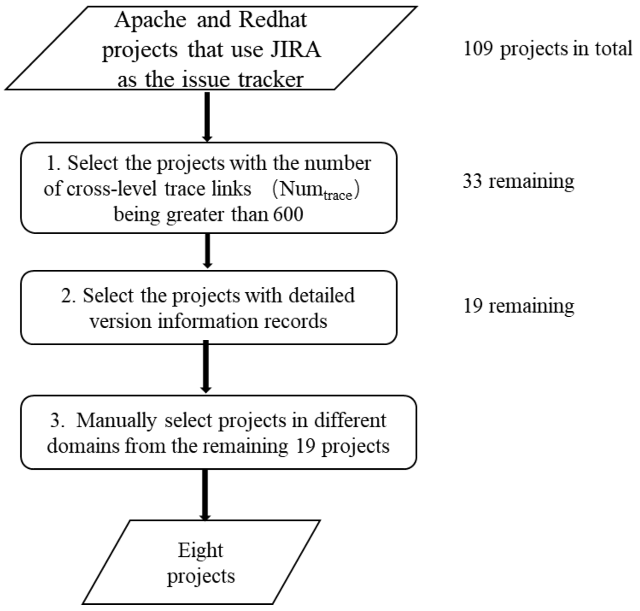

Apache and Redhat projects that use JIRA as the issue tracker
109 projects in total
1. Select the projects with the number of cross-level trace links (Num$_{trace}$) being greater than 600
33 remaining
2. Select the projects with detailed version information records
19 remaining
3. Manually select projects in different domains from the remaining 19 projects
Eight projects

**Figure 8.** Flowchart of selecting projects for this study.

**Table 2.** Details of selected projects.

| Project | Affiliated Foundation | Number of Requirements | Number of Cross-Level Trace Links | Project Duration (till 10 March 2022) | Related Domain |
|---|---|---|---|---|---|
| BEAM | Apache | 1837 | 1637 | Over 6 years | Data stream processing |
| CB | | 1985 | 1814 | Over 10 years | Mobile software development |
| FH | Redhat | 1498 | 1906 | Over 7 years | Enterprise mobile application platform |
| JBIDE | | 4464 | 4208 | Over 18 years | Development tool integration |
| AAH | | 482 | 620 | Over 3 years | Application configuration and deployment platform |
| KEYCLOAK | | 2978 | 3788 | Over 9 years | Authentication and rights management |
| KOGITO | | 2244 | 2998 | Over 5 years | Service process modeling |
| PROJQUAY | | 483 | 647 | Over 4 years | Openshift container platform |

**Data collection.** For each requirement entry, we collected requirement ID, text information, and process information, and other metadata fields (e.g., Section 6.1), including *type, summary, description, labels, components, creator, assignee*, and *create_time* (creation time).

All requirement texts (including text descriptions of historical requirements and new requirements) of the project were used to construct a corpus to pre-train the BERT model. For making the constructed dataset more suitable for practical scenarios, a released version from each project was selected, and the requirements before and after the release time were refined as historical and new requirements, respectively. The target splitting point should satisfy a condition: after the project dataset is split with this version, the ratio of trace links in the training set to those in the test set is between 2:1 and 4:1.

Table 3 lists the information, such as the training set, test set, and specific dataset size of each project, the ratio of the trace links in the training set to those in the test set, and the splitting version. The trace space is the product of the numbers of HLRs and LLRs [44].

**Table 3.** Dataset size of each project in the DRAFT verification experiment.

| Project | Pre-trained Corpus Size (MB) | Training Set Size | | Test Set Size | | Tracetrain/Tracetest | Splitting Version V |
|---|---|---|---|---|---|---|---|
| | | Trace Space | Trace Links | Trace Space | Trace Links | | |
| AAH | 0.56 | 18,303 | 492 | 5060 | 128 | 3.8: 1 | "4.4.0" |
| PROJQUAY | 1.0 | 18,842 | 487 | 6832 | 160 | 3.0:1 | "quay-v3.5.5" |
| FH | 1.2 | 145,338 | 1285 | 205,038 | 621 | 2.1: 1 | "ios-swift-6.0.0" |
| CB | 3.45 | 270,184 | 1350 | 200,796 | 464 | 2.9:1 | "3.5.0" |
| KOGITO | 3.34 | 280,191 | 2329 | 149,946 | 669 | 3.5:1 | "1.7.0.Final" |
| BEAM | 14.5 | 333,796 | 1264 | 16,063 | 373 | 3.4:1 | "2.29.0" |
| KEYCLOAK | 7.07 | 810,648 | 2943 | 601,183 | 845 | 3.5:1 | "11.0.0" |
| JBIDE | 15.9 | 1,588,846 | 3448 | 853,117 | 850 | 4.0:1 | "4.5.0.Final" |

## *8.3. Implementation and Results of Experimental Evaluation*

### 8.3.1. Baseline Methods and Evaluation Indicators

Four baseline methods, namely classic information retrieval algorithms VSM [14] and LSI [15], relevance feedback technique RF [18], and TraceBERT [26], were selected. The primary reason for selecting VSM and LSI is that they can be used to calculate the text similarity between artifacts in a lightweight and intuitive manner and exhibit a good practical effect. The relevance feedback method is an improvement of VSM technology and can fully use historical trace links and optimize the text vector of the query statements. Thus, this method can achieve superior results to those of VSM when retrieving trace links [15,16]. We used TraceBERT as a baseline algorithm for two reasons. First, in this method, the online negative sampling strategy is used to solve the overfitting problem that is prone to occur when deep learning technology is applied to the field of requirement trace. Second, the algorithm improves the effect of trace link identification through domain knowledge transfer. Experiments have revealed that this method can produce results superior to those of information retrieval and other technologies (including recall and precision).

When applying RF to the identification of new trace links, the query vector of HLRs can be improved based on the trace links recorded in the system. When implementing the TraceBERT method, the second-phase pre-trained model provided by Ref. [26] was used to fine-tune the training set in Table 3 and then evaluate its performance in cross-level requirement trace tasks for each project on the test set.

We used F1 and F2 scores, which are the harmonic mean of precision and recall, as the evaluation indicators. They measure the ability of the algorithm to consider both precision and recall. These measures can be calculated using Equations (14) and (15). For a given HLR set and LLR set, $T$ is used to represent the set of real trace links between them, $C$ to represent the set of candidate trace links identified by the algorithm, and $C_r$ and $C_w$ to represent the correct-link set and incorrect-link set in the candidate link set, respectively. Thus, $C = C_r + C_w$. Next, we have the following expression:

$$Recall = \frac{C_r}{T}, Precision = \frac{C_r}{C} \tag{15}$$

F1 and F2 scores are the harmonic mean of recall and precision:

$$F_\beta = \frac{(1+\beta^2)\cdot Precision\cdot Recall}{\beta^2\cdot Precision + Recall} \tag{16}$$

In the F1 score, the weight of precision and recall is the same, that is, $\beta = 1$. The weight of the recall rate in the F2 score is two times precision. The F2 score is used because a more complete but less accurate set of candidate trace links is more useful when assisting analysts to construct or update trace links. In this case, analysts only manually filter incorrect data from them. If the recall rate is insufficient, analysts identify a small number of requirement pairs with trace links from numerous requirement pairs, which requires considerable time and effort.

### 8.3.2. Evaluation Results and Analysis

#### RQ1: Performances of DRAFT (the Proposed Method) and Baseline Methods in Trace Link Identification

The experimental results are presented in Table 4. The F2 score of DRAFT (the proposed method) is close to or exceeds 80% in five projects, and it is higher than 60% in the other three projects; its F1 score achieves 70–80% in five projects and 50–65% in the other three projects. The proposed method can achieve average F1 and F2 scores of 69.3% and 76.9%, respectively. The results revealed that DRAFT can identify high-quality trace link sets and provide automatic assistance for analysts.

**Table 4.** Comparison of cross-level requirement trace link identification performance between DRAFT and baseline methods on eight datasets.

| **Project/Method** | **DRAFT** | | **VSM** | | **LSI** | | **RF-Keydim** | | **T-BERT** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **F1** | **F2** | **F1** | **F2** | **F1** | **F2** | **F1** | **F2** | **F1** | **F2** |
| AAH | **0.536** | **0.621** | 0.309 | 0.322 | 0.232 | 0.289 | 0.356 | 0.369 | 0.212 | 0.173 |
| PROJQUAY | **0.767** | **0.829** | 0.349 | 0.323 | 0.203 | 0.198 | 0.454 | 0.413 | 0.335 | 0.419 |
| FH | **0.797** | **0.872** | 0.363 | 0.329 | 0.236 | 0.249 | 0.59 | 0.615 | 0.322 | 0.286 |
| CB | **0.611** | **0.660** | 0.486 | 0.545 | 0.505 | 0.460 | 0.589 | 0.574 | 0.503 | 0.460 |
| KOGITO | 0.544 | **0.647** | 0.413 | 0.407 | 0.334 | 0.312 | **0.557** | 0.542 | 0.383 | 0.336 |
| BEAM | **0.739** | **0.796** | 0.540 | 0.513 | 0.423 | 0.411 | 0.67 | 0.704 | 0.471 | 0.556 |
| KEYCLOAK | **0.805** | **0.859** | 0.466 | 0.456 | 0.398 | 0.417 | 0.566 | 0.553 | 0.435 | 0.457 |
| JBIDE | **0.748** | **0.868** | 0.497 | 0.653 | 0.464 | 0.620 | 0.438 | 0.596 | 0.737 | 0.763 |
| AVERAGE | **0.693** | **0.769** | 0.428 | 0.444 | 0.349 | 0.37 | 0.528 | 0.546 | 0.425 | 0.431 |

Additionally, the pair-wise Wilcoxon signed-rank test was used to test the significant difference between the results obtained by DRAFT and the four baseline methods and set two *p*-value thresholds: 5% and 1%. The results are presented in Table 5. DRAFT achieved the highest F2 score in all the eight projects of different scales. Its performance is significantly improved by 32% (*p*-value < 0.01) and 40% (*p*-value < 0.01), compared with the baseline methods VSM and LSI, respectively. In the FH project, the F2 score of DRAFT was up to 62.3% higher than that of the LSI method. The similarity in the text description and semantic information of cross-level requirements is the primary basis for determining the trace link between the two requirements. Conventional VSM and LSI are more intuitive and can be used to calculate the similarity by capturing the common vocabulary between cross-level requirements. However, semantic information (e.g., inability to handle polysemy and close/synonymous words) is not considered. Compared with the baseline methods, in DRAFT, the BERT language model is used as a text feature embedding module. Therefore, the model can capture the implicit semantic association between words when calculating the semantic similarity of the required text [27]. In DRAFT, the language model obtained after the second-phase pre-training on the project corpus is used. Therefore, DRAFT can be adapted based on the project context, which leads to a performance superior to those of information retrieval methods such as VSM and LSI.

**Table 5.** Pair-wise Wilcoxon signed-rank test results of the proposed method and baseline methods.

| **Methods for Comparison** / **Metrics** | **VSM** | **LSI** | **RF-Keydim** | **T-BERT** |
|---|---|---|---|---|
| F1 | $p < 0.01$ | $p < 0.01$ | $p < 0.05$ | $p < 0.01$ |
| F2 | $p < 0.01$ | $p < 0.01$ | $p < 0.01$ | $p < 0.01$ |

$p < 0.05$ indicates significant difference; $p < 0.01$ indicates more significant difference.

The RF method improves the query statement vector of information retrieval based on historical trace links. In TraceBERT, domain knowledge transfer is realized by using the pre-training model, which enhances the semantic expression ability of the model, and its trace effect is improved compared with those of VSM and LSI. TraceBERT achieves an F2 score higher than 0.763 in the JBIDE project, of which the training data were the largest. DRAFT considerably outperformed RF and TraceBERT, with an F2 improvement of 22% ($p$-value < 0.01) and 33% ($p$-value < 0.01), respectively, because heuristic features based on metadata, in addition to the similarity in text semantics, should be considered during the creation of trace links. For example, in some requirement pairs (e.g., between AAH-1074 and AAH-1138, and between JBIDE-26680 and JBIDE-26652), capturing the similarity from the text semantics alone is difficult because of differences in the use of terms. However, the process features have some common properties (the same creator and component). Therefore, DRAFT can identify trace links, but the baseline methods miss them.

In terms of F1, the average improvement of the proposed method compared with the four baseline methods was 0.265 ($p$-value < 0.01), 0.344 ($p$-value < 0.01), 0.165 ($p$-value < 0.05), and 0.268 ($p$-value < 0.01). DRAFT achieved the highest F1 in seven projects. DRAFT was only slightly lower (1.3% lower) than the RF-keydim method in the KOGITO project. The highest F1 and F2 scores in each project are presented in bold in the table. DRAFT presents obvious superiority over VSM, LSI, RF-keydim, and T-BERT.

RQ1 can be answered as follows: DRAFT can identify high-quality cross-level trace link sets for new requirements, and its performance is significantly superior to those of the four baseline methods.

RQ2: Ablation Experiment

1. Role of Second-Phase Pre-Training in DRAFT

Table 6 presents the trace link identification results when the BERT pre-training model provided by Google and the model after second-phase pre-training are used for extracting the natural language description features of requirements. The improvements and gains (percentage of improvements) are displayed in the last two columns of the table.

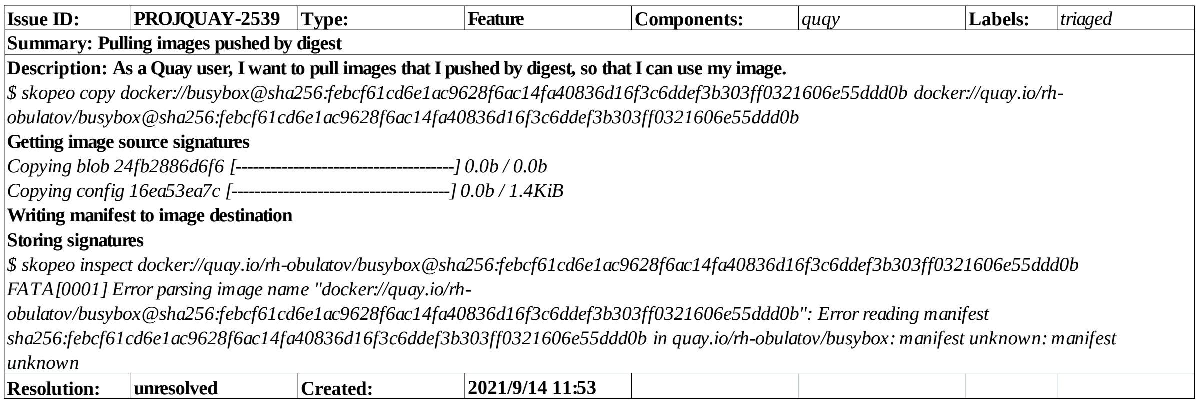
**Table 6.** Effect of second-phase pre-training on trace link identification.

| Project | BERT | | | | | | | | F1 Improvement (Gain) | F2 Improvement (Gain) |
|---|---|---|---|---|---|---|---|---|---|---|
| | Original | | | | Project-Specific Second-Phase Pre-Training | | | | | |
| | P * | R * | F1 | F2 | P | R | F1 | F2 | | |
| AAH | 0.408 | 0.695 | 0.514 | 0.609 | **0.436** | 0.695 | **0.536** | **0.621** | **0.02 (4%)** | **0.01 (2%)** |
| PROJQUAY | **0.826** | 0.744 | **0.783** | 0.759 | 0.685 | **0.875** | 0.767 | **0.829** | −0.02 (−3%) | **0.07 (9%)** |
| FH | **0.598** | **0.703** | **0.646** | **0.679** | 0.543 | 0.698 | 0.611 | 0.660 | −0.03 (−5%) | −0.02 (−3%) |
| CB | **0.778** | 0.747 | 0.762 | 0.753 | 0.696 | **0.931** | **0.797** | **0.872** | **0.04 (5%)** | **0.12 (16%)** |
| KOGITO | 0.427 | **0.745** | 0.543 | 0.648 | **0.430** | 0.740 | **0.544** | 0.648 | 0.00 (0%) | 0.00 |
| BEAM | 0.400 | 0.802 | 0.578 | 0.668 | **0.660** | **0.839** | **0.739** | **0.796** | **0.16 (28%)** | **0.13 (19%)** |
| KEYCLOAK | **0.742** | 0.803 | 0.771 | 0.790 | 0.729 | **0.899** | **0.805** | **0.859** | **0.03 (4%)** | **0.07 (9%)** |
| JBIDE | 0.550 | 0.971 | 0.702 | 0.842 | **0.608** | **0.972** | **0.748** | **0.868** | **0.05 (7%)** | **0.03 (4%)** |
| AVERAGE | 0.591 | 0.776 | 0.662 | 0.719 | **0.598** | **0.831** | **0.693** | **0.769** | **0.03 (5%)** | **0.05 (7%)** |

* P: precision; * R: recall.

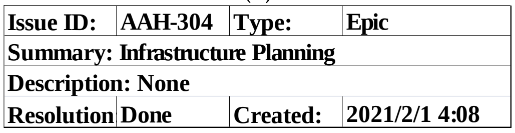
The results of eight projects reveal that second-phase pre-training improves the performance in identifying cross-level trace links, and the average F1 and F2 scores increased by 0.03 (5%) and 0.05 (7%), respectively. Here, F1 and F2 scores are improved after second-phase pre-training in five projects. Their improvements (F1: 16%; F2: 13%) are the most obvious in the BEAM project. In the PROJQUAY project, the F2 score improves by 0.07, but the F1 score decreases slightly (−2%). In the FH project, the performance is similar to that of the original pre-training model, but F1 and F2 scores are slightly reduced after the second-phase pre-training (−3%, −2%).

The dataset size of each project in Table 3 reveals that in projects with small requirement sets, such as AAH, PROJQUAY, and FH, the effect of second-phase pre-training is not as obvious as that in other projects because the pre-training effect of the language model is directly related to the size and quality of the corpus. BooksCorpus (Zhu et al., 2015) and English Wikipedia are used for BERT pre-training. BooksCorpus contains 800 million words, whereas English Wikipedia contains 2500 million words. In both corpora, grammatically standardized natural languages are used. In second-phase pre-training, the requirement documents of the project were used to develop an expected corpus, of which the size is small. For convenience, open-source projects do not have restrictions on the standardization of the text description of the requirement issue log. Therefore, the quality of the corpus is poor. Figure 9 displays two examples of requirement descriptions that are less standard. In Figure 9a), the requirement description contains considerable debugging information; in Figure 9b), the *summary* field is too short, and the *description* field has no available text content. Therefore, a limited improvement is achieved in the design of the requirement trace link identification, and slightly worse results were obtained on a small number of datasets.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **Issue ID:** | **PROJQUAY-2539** | **Type:** | **Feature** | **Components:** | *quqy* | **Labels:** | *triaged* |
| **Summary: Pulling images pushed by digest** | | | | | | | |
| **Description: As a Quay user, I want to pull images that I pushed by digest, so that I can use my image.**<br>*$ skopeo copy docker://busybox@sha256:febcf61cd6e1ac9628f6ac14fa40836d16f3c6ddef3b303ff0321606e55ddd0b docker://quay.io/rh-obulatov/busybox@sha256:febcf61cd6e1ac9628f6ac14fa40836d16f3c6ddef3b303ff0321606e55ddd0b*<br>**Getting image source signatures**<br>*Copying blob 24fb2886d6f6 [-------------------------------------] 0.0b / 0.0b*<br>*Copying config 16ea53ea7c [-------------------------------------] 0.0b / 1.4KiB*<br>**Writing manifest to image destination**<br>**Storing signatures**<br>*$ skopeo inspect docker://quay.io/rh-obulatov/busybox@sha256:febcf61cd6e1ac9628f6ac14fa40836d16f3c6ddef3b303ff0321606e55ddd0b*<br>*FATA[0001] Error parsing image name "docker://quay.io/rh-obulatov/busybox@sha256:febcf61cd6e1ac9628f6ac14fa40836d16f3c6ddef3b303ff0321606e55ddd0b": Error reading manifest sha256:febcf61cd6e1ac9628f6ac14fa40836d16f3c6ddef3b303ff0321606e55ddd0b in quay.io/rh-obulatov/busybox: manifest unknown: manifest unknown* | | | | | | | |
| **Resolution:** | **unresolved** | **Created:** | **2021/9/14 11:53** | | | | |

(**a**)



| | | | |
|---|---|---|---|
| **Issue ID:** | **AAH-304** | **Type:** | **Epic** |
| **Summary: Infrastructure Planning** | | | |
| **Description: None** | | | |
| **Resolution** | **Done** | **Created:** | **2021/2/1 4:08** |

(**b**)

**Figure 9.** Examples of requirement descriptions that are less standard. (**a**) PROJQUAY-2539 with which debugging information is more than natural language text. (**b**) AAH-304 in which the text is too short.

Another reason for the different performance of DARFT is the varied quality of requirement descriptions. In terms of semantics, DRAFT can capture the textual similarities and identify the trace links better for the projects whose traced requirement pairs are described with more consistent terminology usage. For example, when taking the HLR KEYLOACK-7445 ("Test performance of Authorization Services") and LLR KEYLOACK-7620 ("Generating performance datasets for authorization services") as input, DRAFT could yield a high semantic similarity score between their textual description and create their trace link. In contrast, for the projects (e.g., AAH) whose traced requirement pairs share fewer semantically close terminologies, the trace links are more challenging to identify.

Therefore, RQ2 can be answered as follows: second-phase pre-training plays a positive role in trace link identification, and F1 and F2 scores for most projects are improved. However, the performance of cross-level requirement trace link identification for one project degrades slightly.

2. Metadata-Based Heuristic Features in DRAFT

To verify the effect of heuristic features (Section 6), an ablation experiment was designed to compare the effects on trace link identification for new requirements between the case of using text features only and the case of using complete heuristic features. The results are presented in Table 7. Compared with using text features only, DRAFT considerably improves precision, recall, F1 score, and F2 score in all projects when complete heuristic features are used. The average improvements of F1 and F2 scores are 0.274 (72%) and

0.325 (84%), respectively. These two scores improve the most in the PROJQUAY project; their improvements are 0.44 (130%) and 0.50 (150%), respectively.

**Table 7.** Effect of heuristic features on trace link identification.

| Project | Whether to Use Heuristic Features | | | | | | | | F1 Improvement (Gain) | F2 Improvement (Gain) |
|---|---|---|---|---|---|---|---|---|---|---|
| | No | | | | Yes | | | | | |
| | P * | R * | F1 | F2 | P | R | F1 | F2 | | |
| AAH | 0.268 | 0.375 | 0.313 | 0.347 | **0.436** | **0.695** | **0.536** | **0.621** | **0.22 (70%)** | **0.27 (78%)** |
| PROJQUAY | 0.344 | 0.331 | 0.338 | 0.334 | **0.685** | **0.875** | **0.767** | **0.829** | **0.44 (130%)** | **0.50 (150%)** |
| FH | 0.339 | 0.403 | 0.368 | 0.388 | **0.696** | **0.931** | **0.797** | **0.872** | **0.43 (117%)** | **0.48 (124%)** |
| CB | 0.48 | 0.416 | 0.446 | 0.427 | **0.543** | **0.698** | **0.611** | **0.660** | **0.21 (47%)** | **0.23 (54%)** |
| KOGITO | 0.404 | 0.424 | 0.414 | 0.420 | **0.43** | **0.74** | **0.544** | **0.647** | **0.13 (31%)** | **0.23 (55%)** |
| BEAM | 0.402 | 0.374 | 0.387 | 0.379 | **0.66** | **0.839** | **0.739** | **0.796** | **0.35 (90%)** | **0.42 (111%)** |
| KEYCLOAK | 0.45 | 0.433 | 0.441 | 0.436 | **0.729** | **0.899** | **0.805** | **0.859** | **0.36 (82%)** | **0.42 (96%)** |
| JBIDE | 0.572 | 0.912 | 0.703 | 0.815 | **0.608** | **0.972** | **0.748** | **0.868** | **0.05 (7%)** | **0.05 (6%)** |
| **AVERAGE** | 0.407 | 0.459 | 0.426 | 0.443 | **0.598** | **0.831** | **0.693** | **0.769** | **0.274 (72%)** | **0.325 (84%)** |

* P: precision; * R: recall.

The results revealed that these heuristic features extracted from metadata contain important commonalities between cross-level requirements, which provide a basis for the creation of trace links. For example, the find_coms feature in Section 6.1 reflects whether the components involved in a pair of cross-level requirements were identical. If the two features are submitted for the same component, a trace link can exist between them. The commonality in these processes is a supplement of the commonality in the requirement text description. DRAFT comprehensively analyzes the semantic similarity and process information commonality between cross-level requirements and provides a comprehensive and sufficient basis for trace link creation.

RQ3 can be answered as follows: by fusing metadata-based heuristic features, DRAFT can comprehensively capture the commonality in cross-level requirement pairs, which helps to create higher-quality trace links.

## 9. Discussion

### *9.1. Validity Threats*

External validity risks typically originate from the selection of test items and the construction of datasets. To mitigate external validity threats, eight open-source software were selected from various domains. We collected cross-level requirements and their trace links from corresponding JIRA issue log trackers and constructed datasets for experimental evaluation. In addition to JIRA, other widely used issue log trackers, such as Github and Bugzila, are used for requirement acquisition and management. Although the method of storage and requirement management in these methods differs from that in JIRA, the process-related features of the requirements selected in this study (such as the requirement creator) are reflected on these platforms and are all available. Therefore, the proposed method provides a reference for the construction and evolution of cross-level requirements on these platforms.

To eliminate the internal validity threats, we selected the release date of the actual historical version in the project as the splitting point when splitting the training set and the test set to ensure the experimental scenario was as close as possible to the requirement trace practice. We selected four automated trace methods (including information retrieval-based and deep-learning-based methods) from previous studies as baseline methods to reuse open-source codes to avoid implementation errors and ensure accurate experiment execution.

To alleviate the threats of structural validity, we selected the most widely used indicators, such as precision, recall, F1 score, and F2 score, in requirement trace in the experimental evaluation stage. Finally, the trace link identification results can be comprehensively and objectively evaluated in terms of accuracy and completeness.

### *9.2. Limitations*

A second-phase pre-training was performed for the BERT language model on the project requirement text corpus, and a project-specific language model that encodes the semantic knowledge of projects was obtained. When the corpus is of large size and high quality, the pre-training effect of the language model can be maximized. In this study, the data size for the second-phase pre-training is considerably smaller than the data size for the initial pre-training (0.5–15.5 MB vs. 800 M and 2500 M words), and the data are also less standard. The second-phase pre-training still plays a positive role in most projects in this study. In the future, superior results can be obtained by collecting more project-related data (not limited to requirements) and performing fine-tuned preprocessing.

The proposed requirement trace link update method applies to new requirements in the process of project evolution. In practice, in addition to new additions, deletions and modifications of requirements should also be considered. As mentioned in Section 2.2, traceability changes caused by the deletion of requirements are simple, and traceability updates caused by modification of requirements can be performed based on the proposed method. However, no related experiments were conducted. In the future, the comprehensive support of the proposed method should be studied for the three evolution scenarios of cross-level requirement trace.

## 10. Conclusions

During the iteration process of open-source software projects, new requirements are frequently added. Therefore, cross-level trace links should be updated in a timely manner. To address this problem, a cross-level requirement trace method fusing heterogeneous features, that is, DRAFT, was proposed to fully use historical data and abstract the trace link identification method. First, we investigated the project-specific second-phase pre-training method based on BERT. This method can enhance the ability of the pre-training to represent project-related terms and extract the text features in the natural language requirement description that integrates context information. Second, we studied the heuristic feature extraction method for process data to obtain comprehensive feature representations of requirement entries. This method can extract direct features between candidate requirement pairs and extended features based on historical trace list. We then explored the neural network architecture that can fuse heterogeneous features. This architecture can train the requirement trace link identification model, providing automated support for analysts. Finally, we collected cross-level requirements and trace links between them from real-world open-source projects, developed datasets based on the trace link update scenario for new requirements in practical scenarios, and verified the application effect of DRAFT. The experiment results revealed that DRAFT outperformed baseline methods such as VSM, LSI, and TraceBERT in the trace link update task. Although DRAFT is designed to trace requirements at different levels, it can also be referenced to the construction or evolution of trace links between other textual software artifacts (e.g., design documents, UML models, and test cases). Moreover, the architecture of DRAFT can be easily extended to incorporate more features of the related artifacts.

**Author Contributions:** Conceptualization: J.T., L.Z. and X.L.; methodology: J.T.; validation: J.T. and X.L.; formal analysis: J.T. and X.L.; resources: J.T.; data curation: J.T.; writing—original draft: J.T.; writing—review editing: X.L.; supervision: L.Z. and X.L.; project administration: X.L.; funding acquisition: X.L. and L.Z. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work has been funded by the National Science Foundation of China Grant No. 62102014 and 62177003. It is also partially supported by State Key Laboratory of Software Development Environment No. SKLSDE-2021ZX-10.

**Data Availability Statement:** The data that support the findings of this study are available within the article.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Gorschek, T.; Wohlin, C. Requirements Abstraction Model. *Requir. Eng.* **2006**, *11*, 79–101. [CrossRef]
2. Dong, L.; Liu, K.; Zhao, P.; Dai, J. A Quantitative Analysis Method of Itemized Requirement Traceability for Aircraft Development. In *Complex Systems Design & Management*; Krob, D., Li, L., Yao, J., Zhang, H., Zhang, X., Eds.; Springer: Cham, Switzerland, 2021; pp. 89–101. [CrossRef]
3. Special Committee 205 of Radio Technical Commission for Aeronautics. *DO-178C, Software Considerations in Airborne Systems and Equipment Certification*; RTCA: Washington, DC, USA, 2011; pp. 32–42.
4. IEEE Recommended Practice for Software Requirements Specifications. In *IEEE Std 830-1998*; IEEE-SA Standards Board: New York, NY, USA, 2002.
5. McClure, J. CMMI for Development v 1.3 by Mary Beth Chrissis, Mike Konrad and Sandy Shrum. *ACM SIGSOFT Softw. Eng. Notes* **2011**, *36*, 34–35. [CrossRef]
6. Samuel, B.M.; Bala, H.; Daniel, S.L.; Ramesh, V. Deconstructing the Nature of Collaboration in Organizations Open Source Software Development: The Impact of Developer and Task Characteristics. *IEEE Trans. Softw. Eng.* **2022**, *48*, 3969–3987. [CrossRef]
7. Rempel, P.; Mäder, P. Preventing Defects: The Impact of Requirements Traceability Completeness on Software Quality. *IEEE Trans. Softw. Eng.* **2017**, *43*, 777–797. [CrossRef]
8. Arnold, S.; Ricossa, S. Requirements engineering at CISCO. In Proceedings of the Workshop on Requirements Engineering, International Symposium on Software Reliability Engineering, San Jose, CA, USA, 1–4 November 2010.
9. Rahimi, M.; Cleland-Huang, J. Evolving Software Trace Links between Requirements and Source Code. In Proceedings of the 10th International Workshop on Software and Systems Traceability, SST@ICSE 2019, Montreal, QC, Canada, 27 May 2019; Steghöfer, J.-P., Niu, N., Eds.; IEEE/ACM: New York, NY, USA, 2019; p. 12. [CrossRef]
10. Cleland-Huang, J.; Gotel, O.; Hayes, J.H.; Mäder, P.; Zisman, A. Software Traceability: Trends and Future Directions. In Proceedings of the Future of Software Engineering, FOSE 2014, Hyderabad, India, 31 May–7 June 2014; Herbsleb, J.D., Dwyer, M.B., Eds.; ACM: Rochester, NY, USA, 2014; pp. 55–69. [CrossRef]
11. Maro, S.; Anjorin, A.; Wohlrab, R.; Steghöfer, J.-P. Traceability Maintenance: Factors and Guidelines. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, 3–7 September 2016; Lo, D., Apel, S., Khurshid, S., Eds.; ACM: Rochester, NY, USA, 2016; pp. 414–425. [CrossRef]
12. Mäder, P.; Gotel, O. Towards Automated Traceability Maintenance. *J. Syst. Softw.* **2012**, *85*, 2205–2227. [CrossRef] [PubMed]
13. Hübner, P.; Paech, B. Interaction-Based Creation and Maintenance of Continuously Usable Trace Links between Requirements and Source Code. *Empir. Softw. Eng.* **2020**, *25*, 4350–4377. [CrossRef]
14. Salton, G.; Wong, A.; Yang, C.-S. A Vector Space Model for Automatic Indexing. *Commun. ACM.* **1975**, *18*, 613–620. [CrossRef]
15. Lucia, A.D.; Oliveto, R.; Tortora, G. Assessing IR-Based Traceability Recovery Tools through Controlled Experiments. *Empir. Softw. Eng.* **2009**, *14*, 57–92. [CrossRef]
16. Capobianco, G.; Lucia, A.D.; Oliveto, R.; Panichella, A.; Panichella, S. Improving IR-Based Traceability Recovery via Noun-Based Indexing of Software Artifacts. *J. Softw. Evol. Process.* **2013**, *25*, 743–762. [CrossRef]
17. Hayes, J.H.; Dekhtyar, A.; Sundaram, S.K. Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. *IEEE Trans. Softw. Eng.* **2006**, *32*, 4–19. [CrossRef]
18. Rocchio, J.J. Relevance Feedback Information Retrieval. In *The Smart Retrieval System—Experiments in Automatic Document Processing*; Prentice Hall: Hoboken, NJ, USA, 1971.
19. Hayes, J.H.; Dekhtyar, A.; Osborne, J. Improving Requirements Tracing via Information Retrieval. In Proceedings of the IEEE International Conference on Requirements Engineering, Monterey Bay, CA, USA, 12 September 2003.
20. Li, Z.; Chen, M.; Huang, L.; Ng, V. Recovering Traceability Links in Requirements Documents. In Proceedings of the Conference on Computational Natural Language Learning, Beijing, China, 30–31 July 2015.
21. Cleland-Huang, J.; Czauderna, A.; Gibiec, M.; Emenecker, J. A Machine Learning Approach for Tracing Regulatory Codes to Product Specific Requirements. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ICSE 2010, Cape Town, South Africa, 1–8 May 2010; Volume 1.
22. Gervasi, V.; Zowghi, D. Supporting Traceability through Affinity Mining. In Proceedings of the Requirements Engineering Conference, Karlskrona, Sweden, 25–29 August 2014.
23. Rath, M.; Rendall, J.; Guo, J.L.C.; Cleland-Huang, J.; Mäder, P. Traceability in the Wild: Automatically Augmenting Incomplete Trace Links. In Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, 27 May–3 June 2018; Chaudron, M., Crnkovic, I., Chechik, M., Harman, M., Eds.; ACM: Rochester, NY, USA, 2018; pp. 834–845. [CrossRef]
24. Lin, J.; Poudel, A.; Yu, W.; Zeng, Q.; Jiang, M.; Cleland-Huang, J. Enhancing Automated Software Traceability by Transfer Learning from Open-World Data. *arXiv* **2022**. [CrossRef]
25. Guo, J.; Cheng, J.; Cleland-Huang, J. Semantically Enhanced Software Traceability Using Deep Learning Techniques. In Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, 20–28 May 2017; Uchitel, S., Orso, A., Robillard, M.P., Eds.; IEEE/ACM: New York, NY, USA, 2017; pp. 3–14. [CrossRef]
26. Lin, J.; Liu, Y.; Zeng, Q.; Jiang, M.; Cleland-Huang, J. Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models. In Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22–30 May 2021; IEEE: New York, NY, USA, 2021; pp. 324–335. [CrossRef]

27. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. 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, NAACL-HLT 2019, Minneapolis, MN, USA, 2–7 June 2019; Burstein, J., Doran, C., Solorio, T., Eds.; Association for Computational Linguistics: Rochester, NY, USA, 2019; Volume 1 (Long and Short Papers), pp. 4171–4186. [CrossRef]
28. Bhowmik, T.; Do, A.Q. Refinement and Resolution of Just-in-Time Requirements in Open Source Software and a Closer Look into Non-Functional Requirements. *J. Ind. Inf. Integr.* **2019**, *14*, 24–33. [CrossRef]
29. Gupta, A.; Wang, W.; Niu, N.; Savolainen, J. Answering the Requirements Traceability Questions. In Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, 27 May–3 June 2018; Chaudron, M., Crnkovic, I., Chechik, M., Harman, M., Eds.; ACM: Rochester, NY, USA, 2018; pp. 444–445. [CrossRef]
30. Haering, M.; Stanik, C.; Maalej, W. Automatically Matching Bug Reports with Related App Reviews. In Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22–30 May 2021; IEEE: New York, NY, USA, 2021; pp. 970–981. [CrossRef]
31. Zhou, Y.; Tong, Y.; Gu, R.; Gall, H.C. Combining Text Mining and Data Mining for Bug Report Classification. *J. Softw. Evol. Process* **2016**, *28*, 150–176. [CrossRef]
32. Hall, M.A.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; Witten, I.H. The WEKA Data Mining Software: An Update. *SIGKDD Explor.* **2009**, *11*, 10–18. [CrossRef]
33. Guo, J.; Gibiec, M.; Cleland-Huang, J. Tackling the Term-Mismatch Problem in Automated Trace Retrieval. *Empir. Softw. Eng.* **2017**, *22*, 1103–1142. [CrossRef]
34. Guo, J.; Monaikul, N.; Plepel, C.; Cleland-Huang, J. Towards an Intelligent Domain-Specific Traceability Solution. In Proceedings of the ACM/IEEE International Conference on Automated Software Engineering, ASE'14, Vasteras, Sweden, 15–19 September 2014; Crnkovic, I., Chechik, M., Grünbacher, P., Eds.; ACM: Rochester, NY, USA, 2014; pp. 755–766. [CrossRef]
35. D'Ambros, M.; Lanza, M.; Robbes, R. Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison. *Empir. Softw. Eng.* **2012**, *17*, 531–577. [CrossRef]
36. Guo, L.; Ma, Y.; Cukic, B.; Singh, H. Robust Prediction of Fault-Proneness by Random Forests. In Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE 2004), Saint-Malo, Bretagne, France, 2–5 November 2004; IEEE Computer Society: New York, NY, USA, 2004; pp. 417–428. [CrossRef]
37. Jahan, M.S.; Khan, H.U.; Akbar, S.; Farooq, M.U.; Gul, S.; Amjad, A. Bidirectional Language Modeling: A Systematic Literature Review. *Sci. Program* **2021**, *2021*, 6641832. [CrossRef]
38. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention Is All You Need. In Proceedings of the Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; Guyon, I., Luxburg, U., von Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R., Eds.; Springer: Cham, Switzerland, 2017; pp. 5998–6008.
39. Lee, J.; Yoon, W.; Kim, S.; Kim, D.; Kim, S.; So, C.H.; Kang, J. BioBERT: A Pre-Trained Biomedical Language Representation Model for Biomedical Text Mining. *Bioinformatics* **2020**, *36*, 1234–1240. [CrossRef] [PubMed]
40. Araci, D. FinBERT: Financial Sentiment Analysis with Pre-Trained Language Models. *arXiv* **2019**. [CrossRef]
41. Feng, Z.; Guo, D.; Tang, D.; Duan, N.; Feng, X.; Gong, M.; Shou, L.; Qin, B.; Liu, T.; Jiang, D.; et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In Proceedings of the Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16–20 November 2020; Cohn, T., He, Y., Liu, Y., Eds.; Findings of ACL. Association for Computational Linguistics: Rochester, NY, USA, 2020; Volume EMNLP, pp. 1536–1547. [CrossRef]
42. Dong, L.; Zhang, H.; Liu, W.; Weng, Z.; Kuang, H. Semi-Supervised Pre-Processing for Learning-Based Traceability Framework on Real-World Software Projects. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, 14–18 November 2022; Roychoudhury, A., Cadar, C., Kim, M., Eds.; ACM: Rochester, NY, USA, 2022; pp. 570–582. [CrossRef]
43. Reimers, N.; Gurevych, I. Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, 3–7 November 2019; Inui, K., Jiang, J., Ng, V., Wan, X., Eds.; Association for Computational Linguistics: Rochester, NY, USA, 2019; pp. 3980–3990. [CrossRef]
44. Zogaan, W.; Sharma, P.; Mirakhorli, M.; Arnaoudova, V. Datasets from Fifteen Years of Automated Requirements Traceability Research: Current State, Characteristics, and Quality. In Proceedings of the 25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, 4–8 September 2017; Moreira, A., Araújo, J., Hayes, J., Paech, B., Eds.; IEEE Computer Society: New York, NY, USA, 2017; pp. 110–121. [CrossRef]