Enhanced Abbreviation-Expansion Pair Detection for Glossary Term Extraction

Hussein Hasso[a,*], Katharina Großer[b], Iliass Aymaz[a], Hanna Geppert[a], Jan Jürjens[b,c]

[a] *Fraunhofer FKIE, Fraunhoferstraße 20, Wachtberg (Bonn), 53343, Germany*
[b] *University of Koblenz, Universitätsstraße 1, Koblenz, 56070, Germany*
[c] *Fraunhofer ISST, Emil-Figge-Straße 91, Dortmund, 44227, Germany*

Abstract

Context: Providing precise definitions of all project specific terms is a crucial task in requirements engineering. In order to support the glossary building process, many previous tools rely on the assumption that the requirements set has a certain level of quality. Yet, the parallel detection and correction of quality weaknesses in the context of glossary terms is beneficial to requirements definition. **Objective:** In this paper, we focus on detection of uncontrolled usage of abbreviations by identification of abbreviation-expansion pair (AEP) candidates. **Method:** We compare our feature-based approach (ILLOD+) to other similarity measures to detect AEPs and propose how to extend the glossary term extraction (GTE) and synonym clustering with AEP-specific methods. **Results:** It shows that feature-based methods are more accurate for AEPs than syntactic and semantic similarity measures. Experiments with PURE data-sets extended with uncontrolled abbreviations show that ILLOD+ is able to extract abbreviations as well as match their expansions viably in a real-world setting and is well suited to augment previous synonym clusters with clusters that combine AEP candidates. AEP clusters generated with ILLOD+ are generally smaller than those based on syntactic or semantic similarity measures and have a higher recall. **Conclusion:** In this paper, we present ILLOD+, an extended feature-based approach to AEP detection and propose a workflow for its integration to clustering of glossary term candidates to enhance term consolidation in evolving requirements.

Keywords: Glossary Term Extraction, Abbreviation-Expansion Pair Detection, Synonym Detection, Requirements

1. Introduction

One of the goals in requirements engineering is to improve an opaque system comprehension into a complete system specification [1]. Activities related to glossary building support that goal, since glossaries serve to improve the accuracy and understandability of requirements written in natural language [2].

According to the *International Requirements Engineering Board* (IREB) [3], a glossary is a collection of definitions of terms that are relevant in a specific domain. In addition, a glossary frequently contains cross-references, synonyms, homonyms, and abbreviations [3, 4]. Glossaries serve to enrich the requirement texts with additional important information, which ensures that technical terms are used and understood correctly, and supports communication among project participants [5]. The consequent use of a complete and accurate glossary leads to a more consistent language, resulting in coherent structures for the requirements, which in turn enhances automatic analysability [6, 7]. Finally, a glossary can be reused for future projects within the same application domain to facilitate requirements elicitation and analysis [4, 8].

In order to obtain the mentioned benefits, a glossary should be developed early during the requirements elicitation phase and further maintained over the course of the project, which is also compliant to best practices [5, 9]. For various reasons, many projects tend to build their glossary after the requirements elicitation phase [10–12]. However, this complicates the task, since requirements written without the use of a glossary are more likely to contain imprecise or ambiguous wordings. When multiple terms are used to refer to the same meaning (synonyms), denote spe-

*Corresponding author
Email addresses: hussein.hasso@fkie.fraunhofer.de (Hussein Hasso), grosser@uni-koblenz.de (Katharina Großer), iliass.aymaz@fkie.fraunhofer.de (Iliass Aymaz), hanna.geppert@fkie.fraunhofer.de (Hanna Geppert), juerjens@uni-koblenz.de (Jan Jürjens)
URL: http://jan.jurjens.de (Jan Jürjens)

Published version available under https://doi.org/10.1016/j.infsof.2023.107203

cializations (hyponyms), or terms have multiple meanings (homonyms), this presents a major challenge for the identification of glossary terms. Therefore, beforehand, the analyst has to ensure that the terminology is used consistently, e.g., through syntactic or semantic adjustments. This task affects various inter-requirement relations in parallel.

With this paper, we present an approach that encourages the analyst to start with the glossary building, even when the requirements quality still shows weaknesses, and contributes to resolve two tasks in parallel:

(1) quality improvement of the requirements through reduction of lexical variation and

(2) glossary term identification.

In particular, we integrate the detection of abbreviations and their expansions to this workflow.

This paper is an extension of our previous conference paper [13]. First, the detection of abbreviations is refined, to better cope with heterogeneous abbreviation styles—particularly, *lower-cased* abbreviations and *bi-grams* of two consecutive abbreviations. Second, an extended version of our ILLOD algorithm, which checks **I**nitial **L**etters, term **L**engths, character **O**rder, and **D**istribution of abbreviation-expansion pair candidates, is presented—*ILLOD+*. Through recursive calls and a more sophisticated *character distribution analysis*, it achieves higher recall and precision. Third, the preliminary evaluation on the *PROMISE* [14] data-set [15, 16] with only 30 injected uncontrolled abbreviations is extended to the lager *PURE* [17] data-set with 1934 requirements and a data-set of more than 500 potential abbreviations.

2. Problem Definition

Pohl and Rupp [4] describe basic rules for glossary usage to enable beneficial use in requirements engineering, which can be summarized as follows:

a) There is *only one valid centrally managed glossary*, accessible to all involved personnel.

b) Entries in the glossary shall have a *consistent structure* and name the *source* of the terms, as well as possible *synonyms* and *homonyms*.

c) It is *obligatory to exclusively use the terms and definitions as defined* in the glossary.

d) A responsible must *maintain the glossary over the course of the project*, to ensure its consistency and up-to-dateness. Definitions should be agreed upon by all involved stakeholders.

Term consolidation within the written requirements can help to solve issues that originate in disregarding the above rules, such as use of:

1) terms that are not relevant any more (a),c)),

2) changed, incomplete, incorrect, or incomprehensible definitions (a),d)),

3) relevant terms that are not (yet) in the glossary (c)),

4) and uncontrolled synonyms (b),c)).

To facilitate the maintenance task d) and the term consolidation, identification of potential glossary terms within the requirements and identification of synonyms among them can be supported by software tools. We briefly focus on the main problems to be solved by an automated tool for the identification of glossary terms (GTE) [2, 6, 12]. Such a tool needs to solve at least two known problems:

First, since 99% of glossary entries are *noun phrases* (NPs) [18, 19]:

(A) GTE tools need an *accurate noun phrase detection*.

Second, as glossaries deal with domain specific terms and omit duplicates:

(B) GTE tools need to *filter* detected noun phrases to *glossary term candidates*.

Considering (A), *Natural Language Processing* (NLP) pipelines for noun phrase detection, e.g., through *chunking approaches*, are shown to be effective [10, 11]. As such, in this paper we focus on devising an effective technique for (B). Here, *statistical filters* composed of *specificity* and *relevance* measures, as presented by Gemkow et al. [12], could be used, in which beforehand identification of homonyms, synonyms, and different spelling variants among detected noun phrases is expected to have a positive effect on accuracy. Since we explicitly consider requirement sets with such quality weaknesses, we first focus on:

(B1) GTE tools need to *identify and/or merge homonyms, synonyms, hyponyms and different spelling variants* among detected noun phrases.

To detect such relations among domain terms is also beneficial for the building of initial domain models. However, in the following, we focus on the consolidation of terms to reduce language variability—and, thus, ambiguity—of requirements.

A tool to support the term consolidation should be able to show which terms are already defined, as well as the change history of their definitions. Further, it should provide the possibility to add new terms and definitions to the glossary, update definitions, maintain synonym lists and references to homonyms or hyponyms, as well as flag entries as approved and final.

In order to check whether a given pair of terms is synonymous, homonymous, or hypernymous, the underlying concepts themselves must be disambiguated [18], which requires good knowledge about the relevant domain. Therefore, candidate term pairs still have to be confirmed or rejected by the analyst.

For this task recall is more relevant than precision [11] since it can be classified as a *hairy requirements engineering task* [20]: First, not all ambiguities (synonyms, hyponyms, homonyms) are easy to spot for human eyes. Second, a user with knowledge of the domain can decide immediately whether a given term pair is synonymous, homonymous or hypernymous. This means that true positives are difficult to find and false positives are generally easily identified by humans. A tool to support this task should therefore have the highest possible recall. To tackle low precision, term clusters are a suitable method of representation. A cluster of size n can combine $(n(n-1))/2$ term pairs and provides a good solution for making a high number of false positives less costly. However, precision should not be shortchanged, since too large clusters can limit the practicability of the tool and the size of clusters can easily exceed the limits of information amount that humans can process at one time [21].

For example, *REGICE* [11] follows a synonym clustering approach. Yet, only *context-based* (semantic) and *text-based* (syntactic) similarity [22] are considered and *abbreviations* must be cleaned up and defined beforehand. Homonyms and hyponyms are not explicitly addressed, but can be spotted as bycatch during inspection of the synonym clusters. However, for homonyms this is only the case for non-disjoint clusters.

For higher recall in synonym detection, additionally *pattern-based* similarity [22] for *controlled abbreviations* can be applied. It refers to clauses where abbreviations are defined by their corresponding expansions using parentheses or keywords such as *"also known as"*, *"abbreviated"* and *"a.k.a."*, e.g.,

- Common Business Oriented Language **abbreviated** COBOL
- AES **(**Advanced Encryption Standard**)**
- Compression / Decompression, **also known as** Codec

More interesting, however, is an algorithm that also supports to resolve *uncontrolled abbreviations*, which are not defined in place when they are used. Uncontrolled abbreviations in requirements are rather common, especially when requirements elicitation is carried out by different persons (in different organizations) and when guidelines for the use of abbreviations are missing or not followed. Abbreviations can be homonymous by having multiple possible expansions within the same requirements set, as they are predominantly used in a project- or domain-specific context, and new projects regularly come up with new word creations. Thus, simple look-up techniques on predefined lists are not sufficient. This leads us to the next problem statement:

(B1.1) GTE tools need to exploratorily *resolve hitherto unknown abbreviations* in comparison to other terms present in the given text.

Since the abbreviation list is part of the glossary, both should be built in parallel. **The goal is to enable a specific synonym detection optimized for matching of abbreviations with their expansions, which can be integrated to the clustering in glossary term extraction (GTE) tools.**

3. Approach Overview

Fig. 1 illustrates our vision of a workflow to consolidate glossary terms within evolving requirements:

First, term candidates are extracted from the requirements texts and clustered into synonym clusters and AEP groups, matching hitherto undefined abbreviations detected in the texts with all their potential expansions among the term candidates. This process should be automated by tools such as REGICE for clustering and IL-LOD for AEP matching.

In a second step, these clusters are manually inspected to identify true synonyms and consolidate them to a preferred term, as well as confirm abbreviation-expansion pairs. Options shall be discussed with involved stakeholders and final results approved by them.

Third, the responsible for glossary maintenance incorporates the outcome of the previous step to the glossary and its related abbreviation list. This might include the introduction of new glossary terms with their definitions, the addition of terms to synonym lists of other defined terms, changes to terms and/or definitions, and the addition of new entries to the abbreviation list.

Based on the updated glossary, in the last, forth step, the requirements are updated to be consistent with the new definitions and preferred terms in the glossary.

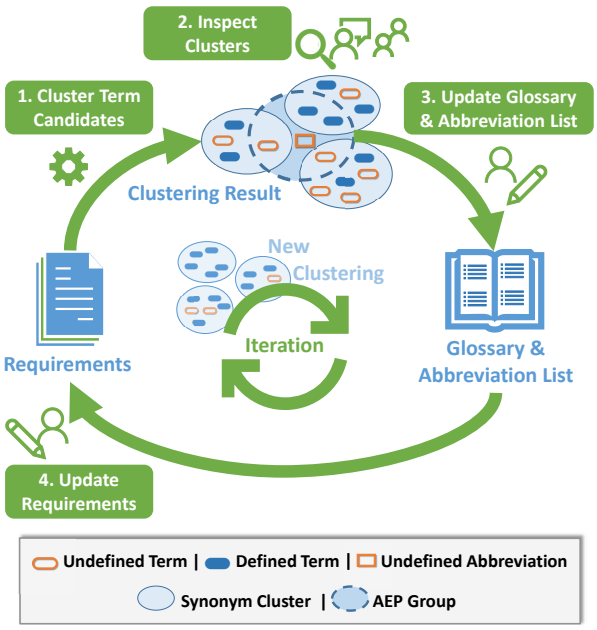

Fig. 1: Envisioned Workflow for Term Consolidation

This process is executed iteratively over the course of the project, reducing the number of undefined terms and abbreviations while the requirements stabilize.

In this paper, we focus on the first step. To enable the automation of detection and clustering of abbreviation-expansion pairs (AEPs), we first compare the accuracy of syntactic and semantic similarity measures with feature-based classification approaches applied to AEPs. In distinction to *semantic* (context-based) and *syntactic* (text-based) similarity measures [22], *feature-based* approaches rely on the evaluation of several distinct syntactic features, e.g., matching of initial letters. We introduce *ILLOD+*, a feature-based binary classifier extending the algorithm of Schwartz and Hearst [23]. It checks **I**nitial **L**etters, term **L**engths, **O**rder, and **D**istribution of characters. Finally, we propose how tools like ILLOD+ can be integrated beneficially into the clustering of glossary term candidates.

4. Related Work

Gemkow et al. [12] reduce the number of glossary term candidates for glossary term extraction (GTE) by using relevance and specificity *filters*. Improving the precision of glossary term extraction like this is important especially for large data-sets. Yet, they do not regulate the possible presence of synonyms and homonyms when determining term frequencies.

Arora et al. [11] argue that *clustering* of glossary term candidates has the advantage to better mitigate false positives from noun phrase detection (A) and to support candidate filtering (B). In addition, their approach provides guidelines on how to tune clustering for a given requirements set, to *detect synonyms* with high accuracy. They conclude that disjoint clusters should be produced in order to keep the workload for term identification low. In Section 7, we look at this from a new perspective and suggest to use different types of clusters for abbreviations as a special type of synonyms.

There are various approaches for the extraction and recognition of *abbreviation-expansion pairs* (AEPs). In addition to *statistical* [24, 25] and *rule-based* methods [26, 27], there are also *machine learning* methods [28, 29]. Many publications deal with biomedical texts and a few, like Park et al. [30], with the field of computer science. Most work assumes that AEPs are predefined in the text via certain patterns and focus their analyses on the surrounding context of the detected abbreviations, which is also the case for Schwartz and Hearst [23]. In our work, we extend the algorithm *findBestLongForm* presented by Schwartz and Hearst [23] to make it applicable for cross-comparisons where an abbreviation and its expansion may occur in different sentences/requirements and are distributed over the given text. We also show that this extension—ILLOD+—can be used beneficially in extraction and identification of requirements glossary terms.

5. Abbreviation Detection

The first step to AEP-matching is the identification of abbreviations. Since "[t]he styling of abbreviations is inconsistent and arbitrary and includes many possible variations" [31], abbreviation extraction is equivocal. Usually it is achieved by finding *single words* that are *relatively short* and have *several capital letters* [23, 32, 33]. This way, not only acronyms are addressed, but also other forms of abbreviations. However, this might not fit lower cased or truncated words.

To solve this task, a simple algorithm detects abbreviations of the former type. It returns *"true"* for a given word w, if the capital letter portion and the word length exceed respectively fall below specified parameter values, otherwise it returns *"false"*. We test this method on a cleaned list of 1786 abbreviation-expansion pairs known from the field of information technology [34][1] with abbreviations of different styles. We reference this

[1] For reproduction, it is included in the supplemental material [35].

list with L, all abbreviations $a \in L$ with A, and all expansions $e \in L$ with E. To identify suitable parameters for word length and the proportion of capital letters, we perform *F1-optimisation* through an exhaustive search on all possible combinations of the two parameters, since false identification of text as an abbreviation can lead to increased workload in the following steps, we believe that precision should not be disregarded. The search is conducted in the range from 0.01 to 1.0 (in 1/100 steps) for the capital letter portion parameter and from 1 to 20 for the word length parameter. The algorithm is once tested on all $a \in A$ and once on all $e \in E$ to obtain false negative and false positive assignments respectively. After optimization, with word-length ≤ 13 and proportion of capital letters ≥ 0.29, we achieve Precision = 0.922, Recall = 0.923, and F1 = 0.922.

For abbreviations formed of lower case letters only, e.g., "env" for "environment", a second algorithm runs after the previous one. Here, a word w is considered to be an abbreviation if a) it consists of only one letter which is not "a" **or** b) it is not in the linguistic dictionary and word-length is $\leq$ the given limit. If w contains special characters like "-" and "\", w is split at these and we check recursively whether at least one of the subwords is a lower case abbreviation. Again, the maximum word-length is determined by *F1-optimisation* in the range from 1 to 20 and the algorithm is once tested on all lower cased $a \in A$ and once on all lower cased words of all $e \in E$ to obtain false negative and false positive assignments respectively. After optimisation with word-length ≤ 6, we achieve Precision = 0.919, Recall = 0.836, and F1 = 0.875.

Finally, we obtain on L for the overall approach **Precision = 0.956**, **Recall = 0.937**, and **F1 = 0.947**.

On full written text, detection is more challenging and further measures must be taken:

a) Apply a filter for numbers like "3.1.4" or "4-12".

b) Apply a stop word filter sorting out stop words that have only one uppercase letter as first letter, e.g., "The", "Any", or "If".

c) Apply the detection also on all *bi-grams*, where both words have to match to find abbreviations, consisting of two consecutive abbreviated terms, e.g., "Pg Dn" for "Page Down". When a bi-gram abbreviation is found, we remove its single parts from the overall results list, if they only appear in the bi-gram, to avoid redundancies.

d) Every word w that directly ends with a full stop, consists only of lowercase letters, and has ≤ 6 letters is also considered to be an abbreviation.

6. Detection of AEP Candidates

For AEP detection, different types of similarity measures are eligible. In a nutshell, words are *semantically* similar if they have the same meaning and *syntactically* similar if they have a similar character sequence [36]. Semantic measures rely on data from large corpora or *semantic nets*—models of terms and their relations, whereas "syntactic measures operate on given words and their characters without any assumption of the language or the meaning of the content" [36]. Finally, *feature-based* similarity rates features that are common to a given pair of words, e.g. the order of certain letters. Below, we compare three different types of classifiers for AEP detection we implemented in *Python* based on these three types of similarity measures.

6.1. AEP Detection with Semantic Similarity Measures

Most methods to semantic similarity need to know queried terms in advance. This applies to knowledge-based methods that rely on lexical databases such as *WordNet* [37] and corpus-based methods such as *Word2vec* [38]. As a result, these methods are not suitable to solve (B1.1). Thus, we chose FastText (FT) [39] as a generic approach and state-of-the-practice technique to assess the suitability of semantic similarity methods for AEP detection. To assign an abbreviation a to a *potential* expansion t in the upcoming evaluation, our simple semantic classifier returns whether

$$cosine\_sim(pe_{FT}(a), pe_{FT}(t)) \geq threshold,$$

where $cosine\_sim$ is the cosine similarity for two vectors x and y, defined as $\frac{x^T y}{\|x\|\|y\|}$, and $pe_{FT}(x)$ stands for phrase embeddings with FastText: $\frac{1}{|x|}\sum_{w \in x} embed_{FT}(w)$. Although the use of fixed thresholds is unusual for similarity measures that provide a ranking, alternatives, e.g., based on percentiles, have drawbacks in the context of the clustering use case we strive for. We discuss this in more detail in Section 8.

6.2. AEP Detection with Syntactic Similarity Measures

The second type of classifier uses syntactic similarity measures between a and t. For this, several measures, as summarized by Gali et al. [36], can be used, like *Levenshtein-Distance* (LD), *Jaro-Winkler-Similarity* (JWS), an extension of *Jaro-Similarity*, and the *Dice-Coefficient* (DC). We do not choose the extended *Damerau–Levenshtein-Distance* as it considers transpositions and plain LD is therefore more sensitive to changes in the sequence of letters. However, the use of syntactic similarity measures to detect AEPs is limited.

Table 1:
Syntactic and semantic similarities between randomly chosen AEPs $(a, e) \in L$ (* indicates distance measures d^* normalized to similarity in $[0, 1]$ by $d^* = 1 - (d(a, e)/max(|a|, |e|))$ [36]); LD: Levenshtein-Distance, JWS: Jaro-Winkler-Similarity, DC: Dice-Coefficient, FT: FastText

Abbreviation-Expansion Pair (a, e)	**LD***	**JWS**	**DC**	**FT**
(LED monitor, Ligth-emitting diode)	0.15	0.435	0.818	0.442
(Int, integer)	0.286	0.651	0.667	0.21
(PS/2, Personal System/2)	0.235	0.436	0.444	0.142
(IANA, Internet Assigned Numbers Authority)	0.114	0.612	0.316	0.448
(SMM, System Management Mode)	0.136	0.586	0.307	0.269
(U/L, upload)	0.0	0.0	0.444	0.025
(IAP, Internet access provider)	0.042	0.458	0.375	0.152
(CLNS, connectionless network service)	0.0	0.0	0.471	0.303
(MMC, MultiMediaCard)	0.214	0.603	0.333	0.533
(I/O, input/output)	0.083	0.472	0.6	0.147

Table 2:
Average syntactic similarities for all pairs $(a, e) \in L$ and $(a, \hat{a})$ with $\hat{a} = potAbb(e)$ with and without pre-processing (*pre*) (* indicates distance measures d^* normalized to similarity in $[0, 1]$ by $d^* = 1 - (d(a, x)/max(|a|, |x|))$ [36]); LD: Levenshtein-Distance, JWS: Jaro-Winkler-Similarity, DC: Dice-Coefficient

Compared	**LD***	**JWS**	**DC**	**pre**
(a, e)	0.092	0.309	0.419	no
	0.183	0.637	0.422	yes
$(a, \hat{a})$	0.361	0.422	0.861	no
	0.797	0.896	0.865	yes

Typically, abbreviations contain only a small proportion of the letters of their respective extensions. E.g., the pair ("ISO", "International Organization for Standardization") has only a share of 3/14 common characters compared in lower case. This is also reflected in Table 1, where the similarities between randomly selected pairs from L are rather low.

To overcome this, the matching of an abbreviation a and some possible expansion t can be estimated by creating a *potential* abbreviation $\hat{a} = potAbb(t)$ out of the initial letters of the single words of t. Similarity is then measured between a and $\hat{a}$. This contraction allows to compare a and t on a homogeneous representation level. Table 2 summarizes the average values of the syntactic comparisons between (a, e) as well as $(a, \hat{a})$ for all pairs $(a, e) \in L$, where $\hat{a} = potAbb(e)$ following the just mentioned contraction approach.

Further, we apply *pre-processing* by converting the string into lower case letters, removing punctuation marks and the stop words "for", "and", "of", "in", "via" and "be". Table 2 shows, that pre-processing and contraction have a positive effect for all three examined measures. For (a, e), the average (normalized) Levenshtein-Distance improves by 0.705, average Jaro-Winkler-Similarity by 0.587, and the average Dice Coefficient by 0.446. Thus, with $a^c = preprocess(a)$ and $t^c = preprocess(t)$, the second type of classifiers returns whether

$$syntactic\_sim(a^c, potAbb(t^c)) \geq threshold.$$

Although abbreviations usually have short length—in our dataset L, the average after pre-processing is 3.55—it can be assumed that $\hat{a}$ and a still differ in many cases despite pre-processing. For the Levenshtein-Distance, there is a relative difference of 20.3% in average between $\hat{a}$ and a even after pre-processing, which shows that, as assumed [31], the formation and use of abbreviations in information technology is not subject to fixed guidelines/regulations in practice. Even though the average Jaro-Winkler-Similarity and the average Dice-Coefficient-Similarity are close to their ideal value of 1.0, they are potentially prone to many false positive assignments. We address this assumption in Section 6.4.

6.3. AEP Detection with Feature-Based Classification

The third type of classifier is represented by *ILLOD*, an extension of *findBestLongForm* [23], and its refinement *ILLOD+*. Whether (a, t) is a candidate AEP is decided solely on the basis of features of a and the words in t. Thus, the approaches are feature-based, although each feature is identified using conditional rules.

6.3.1. ILLOD

Algorithm 1 specifies ILLOD in pseudo-code. The method $check\_initial\_letters(a, t)$ examines for all letters in a if they correspond to the initial letters of the words in t. Thus, the calls in lines 2 and 4 check intuitively if the a is an acronym of t, but have difficulties

Algorithm 1: ILLOD(a, t)

1 a^c = *preprocess*(a); t^c = *preprocess*(t);
2 **if** *check_initial_letters*(a, t) **then**
3 | return True ;
4 **else if** *check_initial_letters*(a^c, t^c) **then**
5 | return True ;
6 **else if** *check_order*(a^c, t^c) ***and***
compare_lengths(a^c, t^c) ***and***
check_distribution(a^c, t^c) **then**
7 | return True ;
8 **else**
9 | return False ;

with pairs like ("QnA", "Questions and Answers"). To solve this, in line 6 additional features are evaluated:

check_order(a, t) examines if the order of the letters in a can also be found in t and if the initial letters of a and t correspond. We compare the letters in backward reading direction to favour an even distribution over the words of the expansion [23].

compare_lengths(a, t) checks whether the length (count of letters) of a is ≥ the number of words in t. This sorts out pairs like ("A", "Advanced Configuration and Power Interface"), based on the assumption that a should reference as many words in t as possible.

check_distribution(a, t) tests if the letters from a, if present in t, are uniformly distributed over the words in t, to sort out pairs like ("SMS", "Systems Network Architecture") or ("PaaS", "Palo Alto Research Center").

6.3.2. ILLOD+

ILLOD+ differs from ILLOD by recursive calls and an extended distribution analysis, for which the string of t is additionally traversed from left to right. Algorithm 2 specifies ILLOD+ in pseudo-code.

After all letters are lowered in line 1, special characters are cleaned up in line 2. Other than in Algorithm 1, stop words are not removed in this step. If the cleanup changes either string, ILLOD+ is called recursively for the strings cleaned of special characters in line 4.

From line 6 to line 10, ILLOD+ checks whether a^{low} is an acronym of t^{low} or whether a^{low} is an acronym of an extended form t^{ex}. Thus, if the initial letters from words in t match the letters from a. In both cases, (a, t) is assumed to be an AEP candidate.

Algorithm 2: ILLOD+(a, t)

1 a^{low}, t^{low} = *lower*(a, t);
2 a^{csc}, t^{csc} = *clear_special_characters*(a^{low}, t^{low});
3 **if** $a^{csc} \neq a$ ***or*** $t^{csc} \neq t$ **then**
4 | **if** *ILLOD+*(a^{csc}, t^{csc}) **then**
5 | | return True ;
6 **if** *check_initial_letters*(a^{low}, t^{low}) **then**
7 | return True ;
8 t^{ex} = *extend_term*(t^{low});
9 **if** *check_initial_letters*(a^{low}, t^{ex}) **then**
10 | return True ;
11 *rtl_ord*, *rtl_pos* = *check_order_rtl*(a^{low}, t^{ex});
12 **if** *rtl_ord* **then**
13 | *ltr_ord*, *ltr_pos* = *check_order_ltr*(a^{low}, t^{ex}) ;
14 | return *cigde*(a^{low}, t^{ex}, *ltr_pos*, *rtl_pos*)
15 return False

Table 3:
Words $w \in t^{low}$ and their Replacement by *extend_term*(t^{low}) in line 8 of ILLOD+ (alternatives separated by "|")

Word w	**Replacement**
to	2to
one | 1	1one
two | 2	2two
three | 3	3three
four | 4	4four
and | &	&and
plus | +	+plus

In t^{ex}, selected words and sub-strings are replaced through extended forms by an extension operation to cover different syntactical variants, mainly to match written numbers in t with digits in a. Table 3 shows the full words that are extended. Further, in line 8, the sub-string "ex" is replaced with "xex" to match abbreviations like "XML" for "Extensible Markup Language".

In line 11, as a prerequisite for further character distribution analysis, ILLOD+ checks if the right-to-left order of the letters in a^{low} can also be found in t^{ex}. Is this the case, the boolean *rtl_ord* is true. In addition, the method *check_order_rtl*() outputs *rtl_pos*—the right-to-left-positions as an integer list. Since the first characters of a^{low} and t^{ex} have to be equal [23], it starts with index 0. The list indicates at which position of t^{ex} a letter of a^{low} is found. E.g., if *rtl_pos*[2] = 15, the third letter of a^{low} is found at the 16^{th} position in t^{ex}. We refer to these lists as character distributions $D_{(a,t)}$ for (a, t).

If there is a valid character distribution, in line 13 *ltr_ord*—the left-to-right-order of the letters of a^{low}

within t^{ex}—is determined. Some examples for valid distributions $D_{(a,t)}$ from *check_order_rtl*(a, t) and *check_order_ltr*(a, t) are:

("ACS", "Access Control System") = [0, 7, 17]

("ACS", "Access Control System") = [0, 1, 4]

Finally, in line 14, $cigde(a, t, ltr\_pos, rtl\_pos)$ checks if a "good" distribution exists within the complete list of all valid character distributions for the term pair (a, t).

Algorithm 3: cigde(a, t, ltr_pos, rtl_pos)

```
1 valid_distr_list =
    detect_all_valid_distr(a, t, ltr_pos, rtl_pos);
2 for distr in valid_distr_list do
3     if check_if_distr_fits(a, t, distr) then
4         return True ;
5 return False
```

As shown in Algorithm 3, for every character in a^{low}, the method *detect_all_valid_distr* uses the values from *ltr_pos* and *rtl_pos* as left and right boundaries respectively, to find all possible positions for it. For example, valid positions for the letter c within the term pair ("acs", "access control system") are 1, 2 and 7, whereas 1 is the left boundary given by its ltr-order and 7 is the right boundary, given by its rtl-order.

A given term pair (a, t) is accepted by ILLOD+ as an AEP candidate, if at least one of its character distributions $D_{(a,t)}$ is accepted, otherwise it is rejected. For this, *check_if_distr_fits* in line 3 checks the following rules:

R1) If a and t have a contiguous sequence of more than 4 letters in common, then $D_{(a,t)}$ is accepted.

R2) If t contains only one word w that is concatenated and can be split into multiple sub-words $w = w_0 \sqcup \ldots \sqcup w_{n-1}$, then rule compliance must be checked again recursively for (a, t^{split}) with $t^{split} = w_0 \sqcup \ldots \sqcup w_{n-1}$. Examples are "electromagnetic", "high-definition", or "OpenOffice".

R3) If t contains only one word w that cannot be split, there are two different variants:

   A) accepts $D_{(a,t)}$ without any further condition.

   B) applies *check_order_rtl*() to check whether the order of the letters of a is also given on the initial letters of the *syllables* of t. E.g., for t = "electromagnetic", with the syllables "elec-tro-mag-net-ic", a may only contain letters ∈ [e, t, m, n, i] in the given order.

R4) For stop words, Rule R5 is optional. As stop words list, we use ("for", "of", "in", "via", "be", "over", "the", "et", "2to", "&and", "+plus").

R5) If t contains multiple words w:

   a) Each word $w \in t$ must be represented by at least one letter in a.

   b) If exactly one letter from a falls on w, it must be the initial letter of w.

   c) If two or more letters from a fall on w and none of them on the initial letter, they must fall on the initial letters of the syllables of w.

R6) If t contains multiple words and at least for one word $w \in t$ *only* the last letter of w is matched, $D_{(a,t)}$ is discarded.

Rule R1 identifies AEPs that have matching subsequences of letters in common, such as ("temp", "temperature") or ("email", "electronic mail").

Rule R2 analyses sub-words in t, to confirm their representation in a. For terms that contain only one word, we implement no further rules, to keep high recall.

Rule R5 provides a frame for the character distribution $D_{(a,t)}$, which checks for each word w in t that w is neither under- nor over-represented in a. Rule R4 softens Rule R5 for stop words, since they are treated very differently in abbreviations.

Finally, Rule R6 discards character distributions that rely on the last letter of a word, as this would result in an unusual letter composition for the abbreviation.

In contrast to ILLOD, ILLOD+ does not explicitly check for the length of a compared to the word count in t, as this is implicitly covered by the distribution rules.

For variant building, we checked where splitting of rules leads to large differences in the experiment results, while at the same time maintaining a recall similar or better as ILLOD and an improved F1. Based on these criteria we selected Rule R3, which has the largest effect on the results. For other rules, result differences are neglectable or it negatively impacts F1 or recall. The two variants A and B of Rule R3 can be seen as a kind of parameter tuning for ILLOD+ with respect to precision and recall, as can be seen from the evaluation data discussed in the subsequent sections.

6.4. Evaluation on a Synthesized Data-Set

We evaluate the classifiers in precision and recall on a data-set synthesized from a plain abbreviation list. Besides the general focus on recall [11, 20], we consider precision, since we do not want the generated term clusters in our envisioned workflow (c.f. Fig. 1) to become too large for humans to lose track of [11].

Table 4:
F1 performance of AEP detection for different α. *Sem* (FT) corresponds to the semantic classifier in Section 6.1, *Syn* corresponds to the different variants of the syntactic classifier in Section 6.2 (LD: Levenshtein-Distance, JWS: Jaro- Winkler-Similarity, DC: Dice-Coefficient) and *Feat* corresponds to the feature-based classifiers (ILLOD, ILLOD+ A/B) in Section 6.3. Best thresholds are given in the *thold* columns. (*indicates normalised *LD*: $LD^*(a,t) = 1 - (LD(a^c, potAbb(t^c))/max(|a^c|, |potAbb(t^c)|))$ [36])

		$\alpha = 4$		$\alpha = 8$		$\alpha = 14$		$\alpha = 28$		$\alpha = 42$	
	Classifier	**F1**	***thold***	**F1**	***thold***	**F1**	***thold***	**F1**	***thold***	**F1**	***thold***
Sem	**FT**	0.642	*0.25*	0.576	*0.28*	0.525	*0.31*	0.453	*0.33*	0.417	*0.33*
Syn	**LD***	0.896	*0.4*	0.859	*0.57*	0.847	*0.56*	0.815	*0.54*	0.791	*0.51*
	JWS	0.877	*0.61*	0.843	*0.72*	0.821	*0.74*	0.776	*0.8*	0.754	*0.81*
	DC	0.877	*0.61*	0.843	*0.72*	0.821	*0.74*	0.776	*0.8*	0.754	*0.81*
Feat	**ILLOD**	0.951	-	0.949	-	0.941	-	0.934	-	0.928	-
	ILLOD+ A	**0.971**	-	**0.97**	-	**0.965**	-	**0.962**	-	**0.956**	-
	ILLOD+ B	0.949	-	0.948	-	0.946	-	0.94	-	0.934	-

Table 5:
Recall (R) of AEP detection for different α. *Sem* (FT) corresponds to the semantic classifier in Section 6.1, *Syn* corresponds to the different variants of the syntactic classifier in Section 6.2 (LD: Levenshtein-Distance, JWS: Jaro- Winkler-Similarity, DC: Dice-Coefficient) and *Feat* corresponds to the feature-based classifiers (ILLOD, ILLOD+ A/B) in Section 6.3. Best thresholds are given in the *thold* columns. (*indicates normalised *LD*: $LD^*(a,t) = 1 - (LD(a^c, potAbb(t^c))/max(|a^c|, |potAbb(t^c)|))$ [36])

		$\alpha = 4$		$\alpha = 8$		$\alpha = 14$		$\alpha = 28$		$\alpha = 42$	
	Classifier	**R**	***thold***	**R**	***thold***	**R**	***thold***	**R**	***thold***	**R**	***thold***
Sem	**FT**	0.596	*0.25*	0.524	*0.28*	0.449	*0.31*	0.398	*0.33*	0.398	*0.33*
Syn	**LD***	0.882	*0.4*	0.781	*0.57*	0.781	*0.56*	0.781	*0.54*	0.781	*0.51*
	JWS	0.872	*0.61*	0.775	*0.72*	0.773	*0.74*	0.759	*0.8*	0.653	*0.81*
	DC	0.872	*0.61*	0.775	*0.72*	0.773	*0.74*	0.759	*0.8*	0.653	*0.81*
Feat	**ILLOD**	0.913	-	0.913	-	0.913	-	0.913	-	0.913	-
	ILLOD+ A	**0.946**	-	**0.946**	-	**0.946**	-	**0.946**	-	**0.946**	-
	ILLOD+ B	0.905	-	0.905	-	0.905	-	0.905	-	0.905	-

Table 6:
Precision (P) of AEP detection for different α. *Sem* (FT) corresponds to the semantic classifier in Section 6.1, *Syn* corresponds to the different variants of the syntactic classifier in Section 6.2 (LD: Levenshtein-Distance, JWS: Jaro- Winkler-Similarity, DC: Dice-Coefficient) and *Feat* corresponds to the feature-based classifiers (ILLOD, ILLOD+ A/B) in Section 6.3. Best thresholds are given in the *thold* columns. (*indicates normalised *LD*: $LD^*(a,t) = 1 - (LD(a^c, potAbb(t^c))/max(|a^c|, |potAbb(t^c)|))$ [36])

		$\alpha = 4$		$\alpha = 8$		$\alpha = 14$		$\alpha = 28$		$\alpha = 42$	
	Classifier	**P**	***thold***	**P**	***thold***	**P**	***thold***	**P**	***thold***	**P**	***thold***
Sem	**FT**	0.694	*0.25*	0.64	*0.28*	0.632	*0.31*	0.527	*0.33*	0.439	*0.33*
Syn	**LD***	0.911	*0.4*	0.955	*0.57*	0.927	*0.56*	0.851	*0.54*	0.8	*0.51*
	JWS	0.883	*0.61*	0.923	*0.72*	0.876	*0.74*	0.794	*0.8*	0.89	*0.81*
	DC	0.883	*0.61*	0.923	*0.72*	0.876	*0.74*	0.794	*0.8*	0.89	*0.81*
Feat	**ILLOD**	0.992	-	0.988	-	0.971	-	0.955	-	0.943	-
	ILLOD+ A	**0.998**	-	0.995	-	0.984	-	**0.977**	-	**0.967**	-
	ILLOD+ B	0.996	-	**0.996**	-	**0.991**	-	**0.977**	-	0.965	-

6.4.1. Experimental Setup

To estimate the accuracy of AEP detection approaches, a data-set D is needed that contains incorrect and correct AEPs. For this purpose, we compiled D as $D = L \cup S$, where L corresponds to the list from Section 5 and S consists of the pairs (a, e) in which a random element e from E was assigned to a given abbreviation a, not matching the real abbreviation of e. To be more formal, the set S can be described as

$S = \{(a, e) \mid a \in A, e \in E, (a, e) \notin L\}$.

While $|L| = 1786$, S grows to $|S| = 2\,710\,125$. S could be reduced by filtering to pairs with identical initial letter. However, since L contains AEPs in which the initial letters differ, this option is discarded. Since we aim to test on a *balanced* data-set, where the proportion of abbreviations among all terms approximately corresponds to that in requirement texts, we test the presented approaches on different $D_\alpha = L \cup S_\alpha$, where $S_\alpha \subset S$ is randomly chosen from S each time, under the condition that $|S_\alpha| = \alpha * |L|$. To obtain an estimate for α, we extract 5702 NPs from 1934 requirements from 13 projects of the *PURE* data-set [17]. To increase the recall, all words (not only words in NPs) are checked by our extraction rules from Section 5. In total, we extract 414 abbreviations and therefore estimate $\alpha = 5702/414 = 13.77$. This allows us to start with $\alpha = 14$ as reference value. are considered. We choose values (4, 8, 14, 28, 42) by multiplying 14 with a factor $\gamma \in (\frac{1}{3}, \frac{1}{2}, 1, 2, 3)$ and rounding to the next bigger even number by $\lceil 14 * \gamma \rceil$. To avoid disadvantages for classifiers based on syntactic and semantic similarity, thresholds are F1-optimised for all α, given as *thold* in Table 4.

6.4.2. Evaluation Results

The results summarized in Table 4-6 show that the FastText-based classifier performs poorly, in both, precision and recall. This might be because a word embedding obtained from FastText can only inaccurately represent a certain word sense if the corresponding abbreviation has multiple expansions with heterogeneous meanings. Classifiers based on syntactic similarity measures have F1-scores between 75 and 90%—on average 82%, but are outperformed by all ILLOD variants, which have between 93 and 97%, with ILLOD+ A being the best. Table 6 and Table 5 show that, in all cases, all ILLOD variants achieve higher precision *and* recall than the syntactic approaches.

For the task at hand, recall is more relevant than precision, as false positives are generally easily identified by humans [11, 20]. Despite the general focus on recall, with the quadratic increase in term pairs to check, a bad precision may influence the practical feasibility of the tool [11]. With increasingly larger α, a slight weakening of the precision for all ILLODs becomes apparent, as can be seen in Table 6. Yet, it remains above 94%, while syntactic classifiers achieve this only in one case (LD*, $\alpha = 8$) and generally show more heterogeneous results for different α. Surprisingly, compared with the results from Table 2, the classifier using the Levenshtein-Distance generally outperforms the Jaro-Winkler-Similarity- and Dice-Coefficient-based ones. Compared to ILLOD+ A, ILLOD+ B achieves a slightly higher precision for the medium α-values (8,14). Yet, with 0.1 and 0.7% difference, this is negligible.

However, the recall of the feature-based classifiers, as shown in Table 5, stays consistently high across all α. Here, clearly the simpler variant A of ILLOD+ outperforms with 94.6% ILLOD (91.3%) and ILLOD+ B (90.5%). Thus, we recommend ILLOD+ A for usage in the envisioned clustering workflow. Yet, in absolute terms, the recall needs to be compared to a *human achievable recall* [20] to judge the effectiveness of the tool. Due to the synthetic nature of the ground truth in this experiment, no such data is available so far. We plan to evaluate this in future research with involvement of human test subjects. However, a first indicator of human performance is the correct recognition and expansion of abbreviations in medical texts, where humans achieve a total accuracy from 74.5% to 88.7% (28.6% for layman without any assistance) [29]. Yet, experimental setup and the domain specific nature of the dataset are not directly comparable to our data.

All of our results can be obtained within the supplemental material [35].

6.5. Evaluation on a Requirements Data-Set

In the following, we evaluate the performance of the different classifiers with full-written text requirements.

6.5.1. Experimental Setup

To evaluate the practicability of ILLOD(+) for the intended use case, we simulate the uncontrolled usage of abbreviations within full-written text requirements from real world projects, as illustratred in Fig. 2.

In preparation for this, terms that occur in at least two requirements within the examined requirements set P are extracted from the test data, which we read in as .csv-files. This way, it is ensured that when substituting one of these terms by an abbreviation in one requirement, its expansion still remains at some other position in the dataset.

Subsequently, possible abbreviations are suggested for these terms by independent persons not involved as

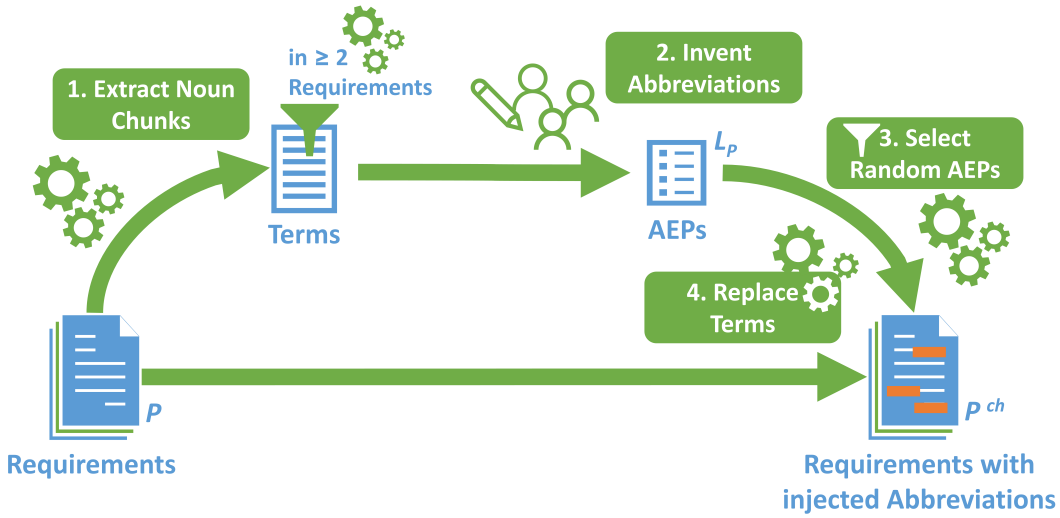

Fig. 2: Process for Requirement Data Set Generation with automatically injected Abbreviations

authors in this work and not familiar with the details of the approach. For this task, we invited research and industry partners.

To simulate the uncontrolled usage of abbreviations, the abbreviations from the AEP list L_P are used to replace their long-form in P and create a changed requirement set P^{ch}. A term t from L_P that appears in n requirements is thereby randomly often replaced with its proposed abbreviation, but at most $n-1$ times to ensure that detection of the corresponding AEP is still possible.

In the preliminary pilot experiment from our previous conference paper [13, 40] with 625 requirements from 15 projects comprised in the *PROMISE* [14] NFR data-set [15, 16], only one person created a list $L_{PROMISE}$ [40] of only 30 suggested abbreviation-expansion pairs and manually replaced the respective long forms in $P_{PROMISE}$ creating $P^{ch}_{PROMISE}$ [40].

For the new evaluation with the larger *PURE* [17] requirements set P_{PURE}, which consists of 1934 requirements from 13 projects, several persons suggested abbreviations for 518 of the total 1338 extracted and cleaned[2] terms. The abbreviations are collected anonymously in a collaborative environment. This way, the abbreviations are more diverse in style and simulate more realistically uncontrolled use in collaborative projects. We reference the list of these 518 AEPs with L_{PURE}. It can be obtained online [35].

It must be noted, that this test data from PURE as well as PROMISE is originally collected from different projects. Yet, we use them in both cases as if they were from one single project, to simulate the AEP detection on a relative large requirement set.

To create several test sets P^{ch}_{PURE} with more abbreviations and different portions α, abbreviations are inserted automatically to P_{PURE}, as illustrated in the right of Fig. 2. An automated routine substitutes a selected number of long-forms from randomly chosen AEPs in L_{PURE} with their short-form. It uses a term-to-IDs-map to ensure that requirements to be changed are found faster. The automated abbreviation injection allows experimentation with different α parameters as well as prevents bias through fixed (manual) selection of abbreviations. In the following, we discuss results for a tenfold random selection of 100 terms to substitute[3], where in each run a new random selction is made. Some further results for other numbers of injections are discussed in Section 9.2 with respect to scalability.

Finally, we apply the extraction approach from Section 5 and the different AEP classification approaches from Section 6 on the different P^{ch} and check whether they can find the inserted abbreviations and assign them to their respective term. Since P_{PURE} already contains 414 abbreviations, we expect the count of abbreviations in P^{ch}_{PURE} to be 514 after injection of 100 abbreviations in each round of the tenfold experiment. Thus, as we extracted in total 5702 NPs from P_{PURE}, we use adjusted thresholds optimized with $\alpha = 11 \approx 5702/514$ for the semantic and syntactic classifiers.

[2] lemmatized with removed determiners

[3] The routine is included in the supplemental material as "*Generate new Test Data with 100 random AEPS from L2*" [35].

6.5.2. Evaluation Results

In the results, assignments of abbreviations to terms are represented by generated AEP groups—clusters of exactly one abbreviation and all its potential expansions. Different sets of AEP groups are generated—one set for each classification approach.

In the pilot experiment from our conference paper, the simple ILLOD creates 115 term tuples, combined to 51 AEP groups for the modified PROMISE data-set $P^{ch}_{PROMISE}$. The extraction detects 29 of 30 inserted abbreviations, which corresponds to a detection recall of $29/30 = 96.6\%$, and ILLOD is able to indicate the correct expansions for 25 of them, which corresponds to an expansion recall of $25/29 = 86.2\%$ and a total recall of $25/30 = 83.3\%$. The other classifiers generate more than twice as many term tuples (AEP candidates) compared to ILLOD in order to indicate the correct expansion for fewer abbreviations—at maximum 22 (total recall: 73.3%). More detailed results can be found online [40].

To further validate these preliminary results, in the extended evaluation with modified PURE data-sets P^{ch}_{PURE}, we investigate some key indicators:

 # missed abbreviations number of inserted abbreviations, that could not be extracted. This value implies **detection recall** [4].

 # found abbreviations number of inserted abbreviations, that have their own AEP group.

 # AEP groups number of generated AEP groups. One AEP group is generated per abbreviation a for which the Boolean "*true*" is returned for at least one term t by the respective classifier.

∅ size of AEP groups Let A^{extr} be the set of abbreviations that we extracted from P^{ch} and k the number for the found abbreviations. The average size is given by $\frac{1}{k}\sum_{a\in A^{extr}}|AEP(a)|$, where $AEP(a)$ references the AEP group for the abbreviation a.

 # matched abbreviations (≙ total recall) number of AEP groups that contain the correct term from L_P.

expansion recall ratio of matched to detected abbreviations,thus *total recall/detection recall*.

cost-effectiveness ratio of matched abbreviations to the average size of all AEP groups.

execution time computing time in seconds to generate the set of AEP groups for the respective classifier.

The results of a first iteration show that the performance of the FastText classifier is weak in terms of both, cost-effectiveness and execution time. It matched only 12 of the 100 inserted abbreviations. With an execution time of 163 seconds the FastText classifier is almost 3 times slower than the second slowest approach, presented by ILLOD+B with 57 seconds. Detailed results for this experiment can be obtained within the supplemental material [35]. To fasten up the next evaluation with ten iterations in a row, we removed the FastText-based one out of the list of classifiers to be evaluated.

The results of the tenfold evaluation are summarised in Table 7. It shows the average results of the ten iterations for all remaining classifiers, which we discuss in the following.

The number of **missed abbreviations** is identical over all approaches (∅ 33.5), as it depends on the initial abbreviation detection algorithm, as defined in Section 5, which is uniformly applied in all cases. Thus, the same abbreviations are missed for all AEP detection approaches. Nevertheless, it can be seen from this, that abbreviation detection recall drops significantly on full-written text requirements—66.5% on average in the tenfold experiment compared to the 93.7% on L, as described in Section 5. This needs to be further investigated for future optimisations and compared to a human achievable recall [20].

The average number of **found abbreviations** remains similar for all approaches. Solely the Levenshtein-Distance-based classifier constantly achieves slightly weaker values than all other classifiers. It should be noted that L_{PURE} contains AEPs with different terms, but represented by the same (homonym) abbreviation. So in some cases, despite the injection of abbreviations from 100 AEPs, the sum of found and missed abbreviations is smaller than 100. As these homonyms are validly created by the independent subjects who contributed to L_{PURE} and it is very likely that such term-clashes for abbreviations occur in a real world setting, too, we did not artificially restrict the injection to disjoint abbreviations. This is the reason why detection recall is calculated based on missed instead of found abbreviations.

The **number of AEP groups** is in all cases larger than the number of 100 injected abbreviations. This is not predominantly caused by false-positives of the abbreviation detection, but mainly due to the 414 abbreviations already contained in the original P_{PURE}. We strive for as few AEP groups as possible, to keep the manual inspection manageable. Yet, also here, the results

[4] #detected abbreviations = #inserted – #missed abbreviations

Table 7:
∅ Results for AEP Detection on P^{ch}_{PURE}, after replacing the terms of 100 randomly selected AEPs from L_{PURE}. *LD, JWS, DC* corresponds to the different syntactic classifier in Section 6.2 (LD: Levenshtein-Distance, JWS: Jaro- Winkler-Similarity, DC: Dice-Coefficient) and *ILLOD, ILLOD+A, ILLOD+B* corresponds to the feature-based classifiers (ILLOD, ILLOD+ A/B) in Section 6.3. Thresholds for syntactic classifiers are F1-optimized for $\alpha = 11 \approx 5702/(414 + 100)$.

AEP Detection	# AEP groups	# found abbreviations	# missed abbreviations	# matched abbreviations	expansion recall	∅ size of AEP groups	cost-effectiveness	execution time (s)
LD	360.6	*62.7*	33.5	26	39.1	21.4	0.842	15.57
JWS	*482.5*	66.0	33.5	55.8	83.9	67.23	1.215	15.43
DC	*454.1*	64.9	33.5	26.5	39.8	34.88	1.342	16.01
ILLOD	375.2	65.7	33.5	58.2	87.5	14.07	0.244	**2.69**
ILLOD+ A	378.7	65.7	33.5	**60.8**	**91.4**	10.73	**0.178**	58.67
ILLOD+ B	376.7	65.7	33.5	39.2	58.9	**8.02**	0.208	68.57

are of similar magnitude, with exception of the Jaro-Winkler-Similarity- and Dice-Coefficient-based classifiers, which produce notably more AEP groups. Generally, the absolute numbers above 360 groups appear too high for realistic manual inspection. However it must be noted, that this corresponds to 514 analysed abbreviations to resolve. As we also discuss in Section 9.2, we do not expect such high numbers to be common in the envisioned iterative application of the approach.

Concerning the **average size of the AEP groups**, we assume smaller groups to be better, again, to keep the manual effort for inspection low. Here, ILLOD+ B creates the on average smallest AEP groups, due to its higher precision. With on average $\approx$ 8 entries, their size is close to the number of elements humans can typically process at once [21]. Yet, ILLOD+ A follows closely with $\approx$ 10, while all other approaches produce significantly larger groups. In particular the Jaro-Winkler-Similarity-based classifier produces extremely large groups (with $\approx$ 67).

In terms of **matched AEPs**, ILLOD+ A, which also achieves the best recall results in Section 6.4, is able to match the most abbreviations with a total recall of 60.8%, closely followed by the simple ILLOD with 58.2%. Yet, in absolute terms, this needs to be compared to a *human achievable recall* [20] to judge the performance of the tool, as already discussed above in Section 6.4. Again, due to the synthetic nature of the ground truth in this experiment, no such data is available and to the best of our knowledge, in the context of requirements engineering, there is no data available to compare our approaches to.

Humans achieve between 74.5 and 88.7% total accuracy in simultaneous recognition and expansion of abbreviations on medical texts if they have domain knowledge or access to a search engine [29]. The weaker performance of our approach is due to relatively low recall for abbreviation detection (see above). Yet, medical text are not requirements and presumably official abbreviations are more recognizable to people with domain knowledge. Abbreviations in our dataset are not domain specific and potentially of more heterogeneous style, as subjects were explicitly asked to not actively research for known standard abbreviations, but create their own suggestions. Further, stakeholders that deal with requirements are not always domain experts. The total accuracy for layman on medical texts without any assistance is with only 28.6% [29] still significantly lower. In addition, the experiments with humans only contain very few abbreviations in each case [29]. Further, accuracy and recall can only be compared if AEP groups are sufficiently small for the correct expansion to be recognised instantly. The figures are thus not really comparable and should at most be seen as indicators. Future work should therefore focus on improving the approach of detecting abbreviations on full written text on the one hand, and on discovering realistic human performance indicators for requirements texts on the other hand.

The **expansion recall** varies from 39.1% for the LD-classifier to 91.4% for the ILLOD+ A classifier, which is followed by the simple ILLOD as second best with 87.5%. ILLOD+ A and ILLOD, both thereby lie in the range of expansion accuracy achieved by humans for medical texts (82.8 to 97.1%) [29].

In combination, ILLOD+ A has on average the best **cost-effectiveness**. All ILLOD variants clearly outperform the syntactic classifier-based approaches in this regard and the simple ILLOD closely follows its extended variants. So overall, consistently with results from Section 6.4, we consider ILLOD+ A the best choice among the examined classifiers.

However, the simple ILLOD has by far the lowest **execution time** of all approaches with around 3 seconds. The approaches based on syntactic classifiers, meanwhile all need around 16 seconds. Both ILLOD+ variants need—with exception of the already discarded FastText-classifier—by far the longest with about 60-70 seconds. All these experiments are conducted on a consumer notebook (Intel i7-10750H, 32GB RAM). We examine execution time because we assume expansion algorithms to be used in tools that align glossary and requirement texts in several iterations, where the number of undefined terms and abbreviations decreases as the requirements stabilise, as shown in Fig. 1. Most frequently, this alignment can be triggered each time after adding or editing requirements. Considering its low execution time and comparably good performance, the simple ILLOD might be the best choice in time critical cases. We briefly discuss scalability in Section 9.2.

Standard deviation σ over all ten iterations indicate that all these averages reasonably represent the actual performances. In addition, the ranking of best and worst for all indications remains the same, also in further ten-fold validation runs. This more detailed data can be obtained from the supplemental material [35].

7. Integration into Clustering Workflow

On the lines of Wang et al. [22], the preceding results confirm that different types of synonyms require different adapted approaches to calculating similarity, in particular for AEP-detection. We experimentally tested different approaches to add AEP candidates to term clusters. Different problems emerged depending on the methods used. E.g., approaches with overlapping clusters have problems to select terms that should be in many different clusters according to a given ground truth. A rationale for this is given in Section 7.2. Other methods that generate clusters with ordinary terms *and* AEP-candidate term tuples, are confusing to humans,

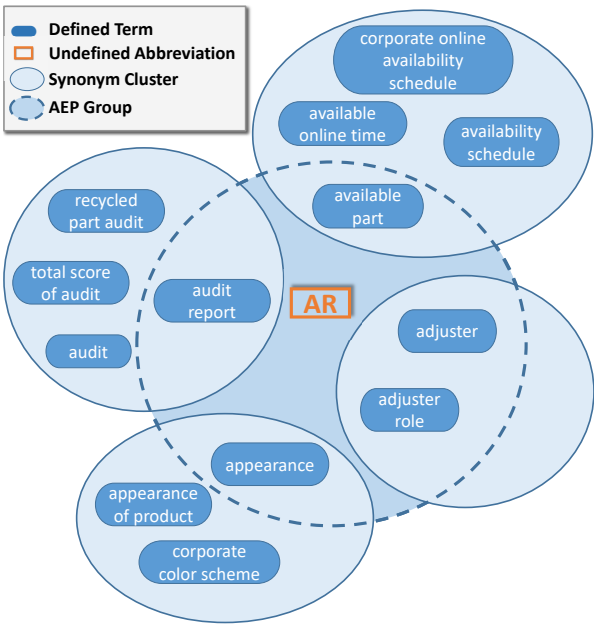

Fig. 3: Glossary term clusters of ordinary terms (light ellipses) and overlay cluster for abbreviation *"AR"* with its possible expansions (dashed circle) for a *"vehicle parts finder system"*.

when the precision of AEP recognition is too low. In the following, we introduce the approach that has the most advantages, as listed in Section 9.3. We do not intend to evaluate different clustering algorithms, but rather to show how two already optimized clustering results—one for ordinary terms according to Arora et al. [11] and one for AEP groups—can be merged.

7.1. Merged Clustering Solution

Arora et al. [11] create their ground truth clusters around a single concept c from the project's domain model, where the clusters also contain variants of terms that are conceptually equivalent to c and terms that are related to c according to the domain model.

Terms within individual AEP groups have a different relation to each other—indicating that two terms can be used as an expansion/definition for the same, as yet undefined, abbreviation. Thus, AEP groups differ in type from the ideal clusters of Arora et al. [11]. As AEP groups are designed to indicate probable ambiguities, they should not be separated in a merged cluster solution, which should be easy to read.

As the ordinary terms within the individual AEP groups do not have to be conceptually related to each other according to the domain model, we must assume that they are distributed over the different clusters of the cluster solution generated for the ordinary terms.

This leads to the conclusion that AEP groups in a merged cluster solution must be considered as so-called overlay clusters, which implies that the AEP groups are included as additional clusters and respective AEP candidates are not inserted into the clusters for the ordinary terms. Fig. 3 shows an example for this, based on the requirements from a *"Vehicle Parts Finder System"* part of the PROMISE data-set [15] as project #5.

7.2. GTE Processing Steps

Considerations from the previous section lead us to propose the approach for the integration of ILLOD+ into a given GTE tool, as outlined in Fig. 4.

First, abbreviations are extracted from the given text, as described in Section 5 and then reduced to only consider yet undefined ones. As an example, assume that A = {"IP", "AR"} was extracted in this step.

Further, general glossary term candidates are extracted through noun chunking, e.g., T = {"audit report", "AP agency", "adjuster role", "automatic AR"} and then cleaned from the abbreviations to a set of ordinary terms, e.g., OT = {"audit report","adjuster role"}.

ILLOD+ is then used to cluster abbreviations with their potential expansions into AEP groups. To continue our example, with α = "AR", we would generate the AEP group G^{α} = {"AR", "audit report", "adjuster role", "automatic AR"}, with T^{α}_{ILLOD+} = {"AR", "audit report", "adjuster role"} and T^{α} = {"automatic AR"}. Afterwards a general synonym clustering approach, such as REGICE [11], is used to cluster the ordinary terms.

As AEP groups are added into the final cluster solution in the last step, this will produce overlapping clusters. To evaluate the solutions generated by this approach, in addition to an ideal cluster solution, a metric is required to determine the score of agreement between an overlapping clustering solution and an overlapping ground truth—the ideal cluster solution. The *OMEGA-Index* Ω, a metric based on pair counts, introduced by Collins et al. [41], can achieve this.

Another argument for generating disjoint clusters of ordinary terms in the second-last step, besides the ones given by Arora et al. [11], is indicated by Ω. It shows the difficulty of making overlapping cluster solutions more similar to the ground truth clustering. For calculation, Ω uses the *contingency table* C. The entries $c_{i,j} \in C$ indicate the number of all pairs that appear in exactly i clusters in the solution and in exactly j clusters in the ground truth. A necessary condition to increase Ω between a generated cluster solution and a given ground truth is to modify the cluster solution so, that their agreement (sum of all diagonal values in C) is increased and their disagreement (sum of all values outside the diagonal) is decreased. Finally, an enlargement of the matrix would cause only a linear increase in the number of agreement fields, while the number of disagreement fields increases quadratically. Therefore, we propose the combination of disjoint clustering with separately calculated AEP group overlay clusters as introduced in Section 7.1.

8. Threats to Validity

In the following, we discuss threats to validity [42] of our abbreviation detection, the ILLOD+ approach to AEP detection, its evaluation, as well as considerations on its integration to glossary term candidate clustering.

Repeatability We provide our source code and data-sets, as well as additional evaluation data [35].

Construct Validity Our approach to glossary building is based on *noun phrases*, as introduced in problem statements (A)/(B). Although, other terms might be defined in a glossary, too, this is the prevalent approach [18, 19]. Threats to the identification of noun phrases are neglectable, as we directly work on extracted terms obtained via well known and reliable NLP techniques, proven for this task [10, 11]. To disambiguate terms based on synonym, homonym, and hyponym detection (B1) and apply this for term consolidation in requirements engineering are well established practices [3, 4, 6, 7, 18]. Towards this more general goal, homonyms and hyponyms are not detected explicitly. Yet, the analyst might be enabled to spot some during manual inspection of the clusters, although in general this problem needs to be addressed in a separate solution. For synonyms in ordinary terms, we assume working solutions from related work [11]. However, focus of this work is on abbreviation-expansion pair detection, as defined in (B1.1). Here, threats to construct validity may apply to the general assumptions on how abbreviations can be recognized and how they are formed from their expansions, since this is manifold [31]. However, threats are neglectable, as parameters for the identification of abbreviations, as well as features for expansion detection, are based on known features [23, 32, 33] retrieved and optimised on *real world examples* that can be adjusted based on larger data-sets or to fit domain specific peculiarities.

Internal Validity To minimize the risk of internal validity threats to our experiments, we tested the similarity measures and classifiers with the same cleaned list of defined abbreviations and under several portions α of abbreviations within the text. Semantic and syntactic similarity measures, are tested with different optimised thresholds. Generally the use of thresholds to

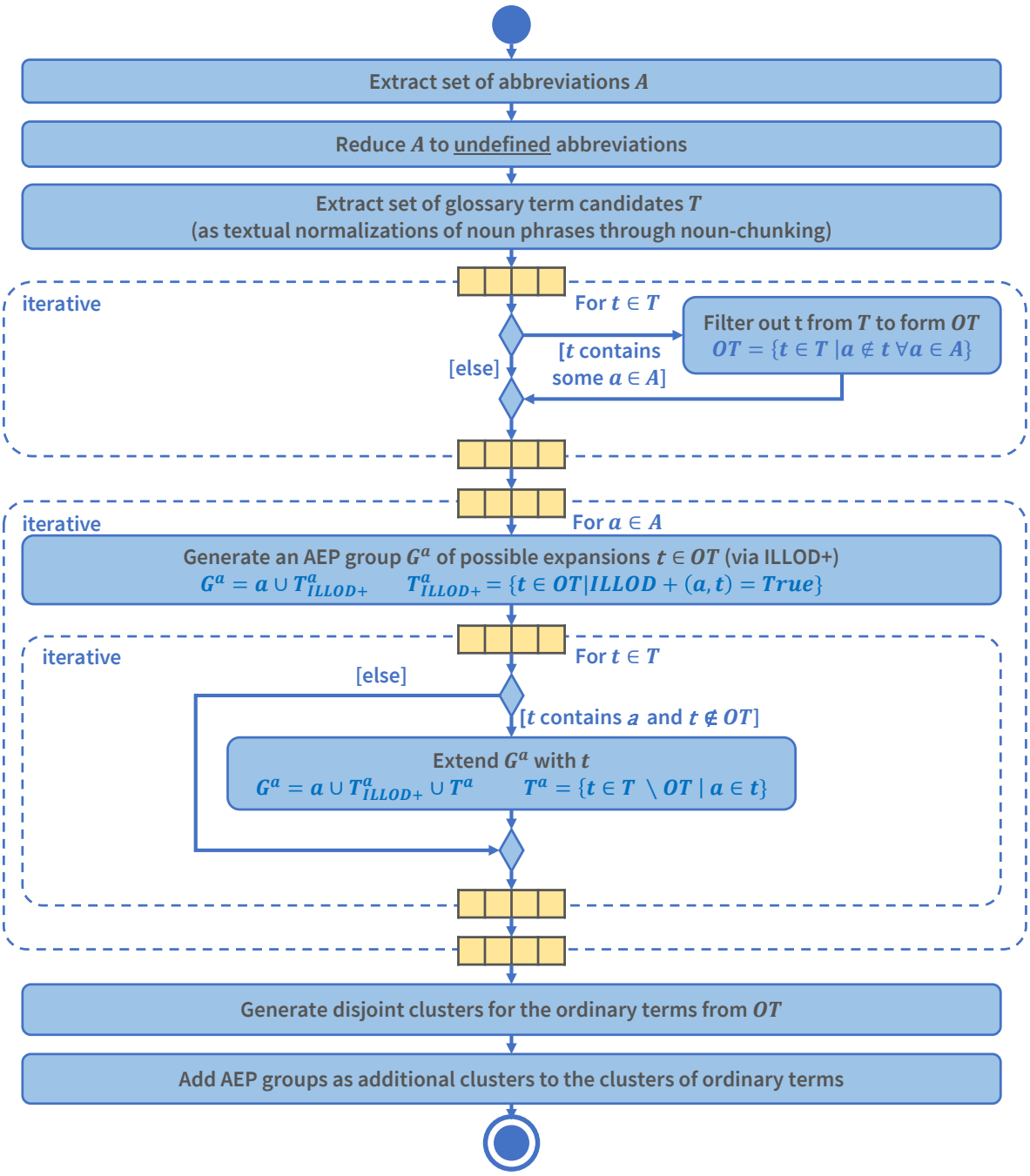

Fig. 4: Proposed Workflow for the Integration of ILLOD+ into Term Clustering of a given GTE Tool

evaluate classifiers that provide a ranking is not common, as it has the drawback of limited adaptability when new term pairs need to be classified. Yet, the presented feature-based classifiers are binary boolean and do not provide a ranking. This needs to be reflected in a proper evaluation approach for comparison—the ranking within the resulting AEP groups is irrelevant. Alternatives to thresholds, e.g., based on percentiles, are more suitable for optimization in terms of precision *or* recall, while we use F1-optimisation, and, more importantly, they have some drawbacks with respect to the clustering approach we aim at. E.g., fixed cutoffs also

fix the size of the AEP group, while flexible cutoffs depend in a similar way on optimisation on the training data, as thresholds, an other means like mean average precision may be suitable to compare the result quality, but would not restrict cluster size at all. Since we want to keep the number and size of clusters manageable and the similarity values in different clusters comparable, we believe that using fixed thresholds for F1 optimisation is an adequate strategy, assumed that the training set (L with $|L| = 1786$ [34]) is large and heterogeneous enough to provide a stable threshold, also for yet unseen term pairs.

External Validity Parameters and features for abbreviation and AEP detection are context and language specific. We discus limitations of our approach in more detail in Section 9. However, the list L [34] we used for optimisation and evaluation is open community built without guidelines and, thus, heterogeneous abbreviation styles not limited to acronyms. Yet, it is domain specific. Further, we only used English terms. Parameters and accuracy might vary for other languages, e.g., in German rules for noun-splitting differ. Meanwhile, parameters can be easily adapted through optimisation on other data-sets. Similar, features evaluated by ILLOD+ can be easily adapted to domain specific patterns. Yet, the tests on the PROMISE and PURE data-sets with requirements from more than ten projects from different domains, indicate some general applicability, but also show that optimisation on real requirements data would be reasonable. We plan to verify and optimise our approach on further data-sets and in a realistic environment in future research.

Conclusion Validity To mitigate threats, all optimisation is conducted on the known list L, while test data is kept separately. For the tests on the synthesized set, test sets are retrieved randomly. For the evaluation on real requirements texts, the modifications to the PROMISE & PURE data-sets are randomly inserted based on abbreviations formed by external independent persons without exposure of details to the authors. The considerations on cluster integration are based on related work [11] and initial experiments on optimization of different clustering algorithms with the OMEGA-Index. However, we plan to substantiate this in future experiments.

9. Discussion

In the following, we discuss the main limitations, scalability and strengths of our approach.

9.1. Limitations of Detection

As can be seen from the precision and recall results in Section 5, our abbreviation detection approach has some limitations due to the heterogeneous nature of abbreviations. To achieve a high precision and recall for lower cased abbreviations is more challenging than for those that have a relative high portion of capital letters. However, lower case abbreviations are in general rare—only 17 of 1786 within L. The only one of those that is not detected by our approach, is "kilo", which is as decimal prefix in the metric system a commonly used word and not in the classical sense an abbreviation of "one thousand". Nevertheless, there are abbreviations that exceed the parameters of our detection. E.g., two examples from the German armed forces to abbreviate a unit of organization and an employment title exceed the limits of our mixed-case detection: First, ("SABCAbwGSchAufg", "Schule ABC-Abwehr und Gesetzliche Schutzaufgaben") is with 15 letters longer than our limit of 13. Second, ("Schirrmstr", "Schirrmeister") has with 0.1 a too low portion of capital letters—this is presumably typical for simply truncated words. We address truncated words only for completely lower cased words, as more common in the English language. It shows that parameters need to be adjusted or some specific rules have to be added for other languages and/or domains with notably different guidelines.

However, detection on full written text is more difficult, as can be seen in Section 6.5 (Table 7), and brings some challenges that require some adaptions or further restrictions for AEP matching. I.a., ILLOD+ rejects abbreviation candidates with length > 15, when the proportion of capital letters is ≥ 0.9. This is necessary, as the list of all valid character distributions grows exponentially for longer words. While generally, such long uppercase abbreviations are rare corner cases within the data examined, the criteria may fit false positives where ordinary terms are written in uppercase letters, e.g., for cooperate design or accentuation. For example, without this restriction, the term "SYSTEM INTEGRATION" extracted from the PURE requirements is falsely classified as an abbreviation bi-gram and causes ILLOD+ to exceed the computation resources of a consumer notebook (Intel i7-7700HQ, 64GB RAM).

The, compared to the high recall in Section 5, rather low numbers of found abbreviations in Section 6.5 reveal the difficulties of abbreviation detection in full written text and with freely invented, heterogeneously styled abbreviations. This shows, that future optimisations need to be based on training data from full text requirements and not beforehand cleaned abbreviation lists like L. To avoid bias, we used the abbreviations

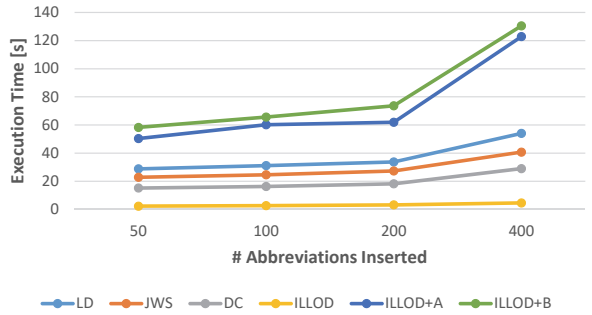

Fig. 5: Execution Time [Seconds] of AEP Detection based on different Classifiers for different Numbers of Inserted Abbreviations to the PURE Requirements. (LD: Levenshtein-Distance, JWS: Jaro-Winkler-Similarity, DC: Dice-Coefficient)

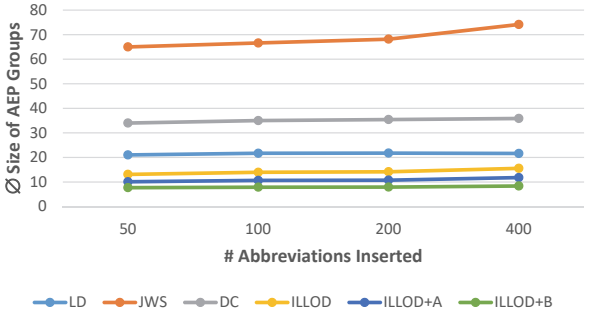

Fig. 6: ∅ Size of AEP Groups for different Classifiers on different Numbers of Inserted Abbreviations to the PURE Requirements. (LD: Levenshtein-Distance, JWS: Jaro- Winkler-Similarity, DC: Dice-Coefficient)

suggested for the terms extracted from the PURE requirements solely for testing. More similar data would be necessary to investigate potential for improvements and establish a benchmark for testing. As a first step, we provide our data-set [35] for replication. However, it is also necessary to evaluate the approach in a realistic environment and compare to human achievable recall [20] in future experiments with stakeholder involvement.

9.2. Scalability

The presented classifiers are not explicitly optimised with respect to time performance. Nevertheless, we discuss some preliminary observations.

As discussed in the previous sub-section, the exponential growth of the distribution map in ILLOD+ can lead to performance issues. However, we assume such long abbreviations to be generally the minority and only scattered within realistic data.

From the tenfold experiment data with 100 injected abbreviations to the PURE requirements in Section 6.5, we learn, that the basic ILLOD approach is the fastest, followed by the syntactic classifiers, while both ILLOD+ variants need the most execution time on a consumer notebook (Intel i7-10750H, 32GB RAM). In experiments on a different machine (consumer notebook, Intel i7-7700HQ, 64GB RAM), the absolute execution times are increased by factor 10, while the ordering and relative difference stay the same. We suspect some dependencies on hardware and virtual machine settings. This needs to be further investigated in a thorough inspection prior to extended time-performance tests.

Further, we conduct the same experiment as described in Section 6.5, again on the same system (Intel i7-10750H, 32GB RAM), but with different numbers of injected abbreviations corresponding to different α. Again, the FastText-based classifier is excluded, as its performance is comparably poor, like described in Section 6.5. Fig. 5 plots the results in execution time. It can be seen, that the order of the approaches is maintained over the different amounts of injected abbreviations. While the simple ILLOD is by a considerable margin the fastest approach, it also maintains a similarly low execution time over all amounts of abbreviations. All other classifiers notably increase in time above 200 injected abbreviations[5]. This effect is most prominent for the ILLOD+ variants. Thus, for time critical applications, the simple ILLOD appears to be the best choice, considering its other performance indicators being comparably similar to its extended versions. However, under consideration that the 400 injected abbreviations add up to a total of more than 800 abbreviations to resolve for the examined data-set, an absolute time of around 2 minutes might be acceptable. In particular, as the envisioned workflow would build up the abbreviation list iteratively as the requirements are written, we do not expect such high numbers to be common. In rarer cases of a posteriori analysis of existing legacy documents, longer execution times might be acceptable by users.

Likewise, the number of AEP groups grows constantly with higher numbers of abbreviations, what is to be expected. As already mentioned in Section 6.5, the high numbers of above 300 observed in the respective experiments and even more above 500 for 200 Injections, are not attractive for a manual inspection approach. However, as stated above, in a realistic use case with the suggested iterative workflow, these cases should be rare. Yet, as the requirements from the PURE data-set are processed as one large test-set, we lack an estimate for realistic amounts in iterative rounds. Fu-

[5]Note that the PURE requirements already originally contain 414 abbreviations (see Section 6.4).

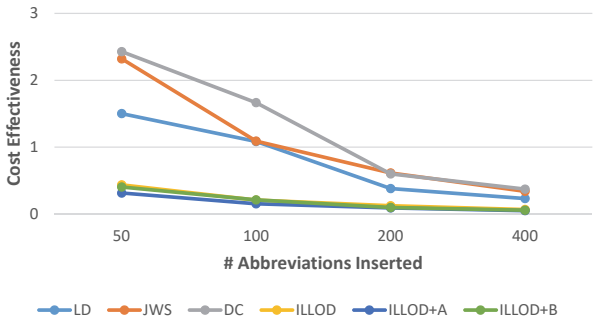

Fig. 7: Cost Effectiveness of AEP Detection based on different Classifiers on different Numbers of Inserted Abbreviations to the PURE Requirements. (LD: Levenshtein-Distance, JWS: Jaro- Winkler-Similarity, DC: Dice-Coefficient)

ture work should cover some case study with user feedback from a real project setting with iterative application. Such an evaluation in a real world project setting with actual user involvement will be necessary to verify the practical relevance of our approach. Further, evaluation on different data sets with different sizes of requirements or respective noun phrases that are not abbreviation would be useful to gain deeper insights to performance scalability.

Nevertheless, the average cluster size, as shown in Fig. 6, remains similar over all injection rates, even on the full large data-set. Here, the ILLOD+ variants provide reasonable sizes with around 8 and 11, closely followed by the simple ILLOD with sizes around 14.

As the number of matched abbreviations increases with the total number of abbreviations, the near to constant average cluster size leads better cost effectiveness for higher injection rates, as shown in Fig. 7. This is in line with the higher precision for lower α observed in Section 6.4.

9.3. Advantages of the Approach

As motivated in Section 1 and 2, generally the use of glossaries can support in all three dimensions to improve an opaque system comprehension into a complete system specification, as defined by Pohl [1]: *content*, *documentation*, and *agreement*. More specific, the outlined process of term consolidation supports the glossary maintenance, as decribed based on Pohl and Rupp [4] in Section 2, addressing the dimensions of *documentation* and *agreement*. Based on our findings, the proposed workflow has the following advantages:

(1) By using AEP groups, we avoid to decide which pair of terms belong together automatically, which is a challenging problem [43].

(2) AEP groups have ergonomic as well as procedural advantages:

   (a) The analyst is motivated to build the list of abbreviations in parallel.

   (b) The analyst has direct insight into how an abbreviation could be expanded alternatively, as alternative expansions are likely to be encountered in the same cluster, and thus the analyst gets another opportunity to reduce ambiguities.

(3) Since the AEP groups are added to the generated cluster of ordinary terms in a post-processing step from a clustering point of view, the AEP groups ensure that unknown abbreviations and proposed expansions are placed in the same cluster, regardless of the clustering algorithm.

(4) Adding additional AEP groups lead to a final result with overlapping clusters, but mitigates the disadvantages of such, as these additional clusters are of different type than those of the ordinary terms.

(5) Using a feature-based approach to AEP detection, as ILLOD+, provides high flexibility to adjust to domain specifics, as new rules can easily be added.

We conducted preliminary experiments with *hybrid* approaches to AEP detection, combining different types of classifiers. E.g., to check the *initial letter equivalence* rule contained in ILLOD in a pre-processing step for all syntactic measures. This leads to increased accuracy for this type of classifier, as can be learned from the detailed evaluation data [35]. However, due to the nature of feature-based approaches of combining and potentially weighting different rules/features, it appears to be more plausible, to potentially integrate syntactic measures as additional rules here, rather than to outsource other features to excessive pre-processing.

Further, we assume our approach not only to be relevant for early harmonization of requirements document terminology, but also if glossary and abbreviation list have to be built over several documents spanning multiple project phases and/or involved organizations and domains, as for distributed specifications, as described, e.g., by Großer et al. [44]. In addition, the different clusters for different synonym types not only help the analyst to select a preferred term, but also to build synonym groups in more advanced glossaries or thesauri with cross references [18] and context specific grouping. The clustering can help to identify such contexts. Further, in such an advanced glossary system, abbreviation and feature-based AEP detection could, e.g. be

integrated into the term search. If a search string is identified as a potential abbreviation, the search can be narrowed to abbreviations or, if it is not yet defined, return potential expansions as search result, which would potentially not be found by a classical syntactic search. We plan to investigate this in currently ongoing research.

Moreover, the feature-based AEP detection through ILLOD+ could be of use for tasks beyond term consolidation for glossaries. For example, tracing approaches that map entities from different models via similarity of element names, as e.g., Peldszus et al. [45, 46] do for data-flow diagrams and source code, could be enhanced to better cope with abbreviations.

10. Conclusions

Early glossary building and synonym detection is relevant to reduce ambiguity in requirements sets, e.g. through definition of preferred terms [18]. We demonstrate that different types of synonyms [18] need different treatments in detection. In particular, classical syntactic and semantic similarity measures perform poorly on abbreviations, as we show with our experiments in Section 6. With our ILLOD+ tool, we present a new feature based approach to AEP detection, which outperforms those classic approaches. It is also more flexible, as rule sets can be easily adapted to context specific characteristics, e.g., guidelines or languages. Initial experiments indicate that investigation of hybrid approaches might be promising, though. We further propose how to integrate groups of abbreviations and their potential expansions to clusters of ordinary glossary term candidates as additional separate type of clusters.

This enables analysts to build the abbreviation list in parallel to the glossary and start this process early already on preliminary requirements. Further, we assume our approach not only to be relevant for early harmonization of requirements document terminology, but also if glossary and abbreviation list have to be built over several documents spanning multiple project phases and/or involved organizations and domains [44]. In addition, different clusters for different synonym types can support the building of synonym groups for glossaries or thesauri with cross references [18] and context specific grouping as well as domain models. The results of our experiments on synthetic datasets are promising and open the door for next step evaluations in realistic environments with practical user feedback.

Acknowledgments

This work is supported by project EnTrust in the research initiative of the Ministry of Science and Health, Rhineland-Palatinate, Germany and European Space Agency's (ESA) NPI program (No. 4000118174/16/-NL/MH/GM). We thank the volunteers who provided us with abbreviation examples for our experiments.

References

[1] K. Pohl, The three dimensions of requirements engineering, in: J. Bubenko, J. Krogstie, O. Pastor, B. Pernici, C. Rolland, A. Sølvberg (Eds.), Seminal Contributions to Information Systems Engineering: 25 Years of CAiSE, Springer, 2013, pp. 63–80. doi:10.1007/978-3-642-36926-1_5.

[2] K. Bhatia, S. Mishra, A. Sharma, Clustering glossary terms extracted from large-sized software requirements using FastText, in: 13th Innovations in Software Engineering Conference, Formerly known as India Software Engineering Conference (ISEC'20), 2020, pp. 1–11. doi:10.1145/3385032.3385039.

[3] M. Glinz, A Glossary of Requirements Engineering Terminology, Tech. rep., International Requirements Engineering Board IREB e.V. (5 2014).

[4] K. Pohl, C. Rupp, Requirements Engineering Fundamentals, 2nd Edition, Rocky Nook, 2015.

[5] A. van Lamsweerde, Requirements engineering, John Wiley & Sons, Inc., 2009.

[6] A. Dwarakanath, R. R. Ramnani, S. Sengupta, Automatic extraction of glossary terms from natural language requirements, in: 21st IEEE International Requirements Engineering Conference (RE'13), IEEE, 2013, pp. 314–319. doi:10.1109/RE.2013.6636736.

[7] Y. Park, R. J. Byrd, B. K. Boguraev, Automatic glossary extraction: Beyond terminology identification., in: 19th International Conference on Computational Linguistics (COLING'02), Vol. 1, 2002, pp. 1–7. doi:10.3115/1072228.1072370.

[8] N. Kiyavitskaya, N. Zeni, L. Mich, D. M. Berry, Requirements for tools for ambiguity identification and measurement in natural language requirements specifications, Requirements engineering 13 (3) (2008) 207–239. doi:10.1007/s00766-008-0063-7.

[9] K. Pohl, Requirements Engineering, Springer, 2010.

[10] C. Arora, M. Sabetzadeh, L. Briand, F. Zimmer, Automated checking of conformance to requirements templates using natural language processing, IEEE Transactions on Software Engineering 41 (10) (2015) 944–968. doi:10.1109/TSE.2015.2428709.

[11] C. Arora, M. Sabetzadeh, L. Briand, F. Zimmer, Automated extraction and clustering of requirements glossary terms, IEEE Transactions on Software Engineering 43 (10) (2017) 918–945. doi:10.1109/TSE.2016.2635134.

[12] T. Gemkow, M. Conzelmann, K. Hartig, A. Vogelsang, Automatic glossary term extraction from large-scale requirements specifications, in: 26th IEEE International Requirements Engineering Conference (RE'18), IEEE, 2018, pp. 412–417. doi:10.1109/RE.2018.00052.

[13] H. Hasso, K. Großer, I. Aymaz, H. Geppert, J. Jürjens, Abbreviation-expansion pair detection for glossary term extraction, in: V. Gervasi, A. Vogelsang (Eds.), Requirements

Engineering: Foundation for Software Quality, Springer International Publishing, 2022, pp. 63–78. doi:10.1007/978-3-030-98464-9_6.

[14] J. Sayyad Shirabad, T. Menzies, PROMISE software engineering repository, School of Information Technology and Engineering, University of Ottawa, Canada (2005).
URL http://promise.site.uottawa.ca/SERepository/

[15] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc, Automated classification of non-functional requirements, Requirements Engineering 12 (2) (2007) 103–120. doi:10.1007/s00766-007-0045-1.
URL http://ctp.di.fct.unl.pt/RE2017/downloads/datasets/nfr.arff

[16] X. Zou, R. Settimi, J. Cleland-Huang, Improving automated requirements trace retrieval: a study of term-based enhancement methods, Empir Software Eng 15 (2) (2009) 119–146. doi:10.1007/s10664-009-9114-z.
URL http://ctp.di.fct.unl.pt/RE2017/downloads/datasets/nfr.arff

[17] A. Ferrari, G. O. Spagnolo, S. Gnesi, Pure: A dataset of public requirements documents, in: 25th IEEE International Requirements Engineering Conference (RE'17), 2017, pp. 502–505. doi:10.1109/RE.2017.29.

[18] ISO, 25964-1: Information and documentation — Thesauri and interoperability with other vocabularies — Part 1: Thesauri for information retrieval, ISO (2011).

[19] J. S. Justeson, S. M. Katz, Technical terminology: some linguistic properties and an algorithm for identification in text, Natural language engineering 1 (1) (1995) 9–27. doi:10.1017/S1351324900000048.

[20] D. M. Berry, Empirical evaluation of tools for hairy requirements engineering tasks, Empirical Software Engineering 26 (6) (2021) 111. doi:10.1007/s10664-021-09986-0.

[21] G. A. Miller, The magical number seven, plus or minus two: Some limits on our capacity for processing information., Psychological Review 63 (2) (1956) 81. doi:10.1037/h0043158.

[22] Y. Wang, I. L. Manotas Gutiérrez, K. Winbladh, H. Fang, Automatic detection of ambiguous terminology for software requirements, in: International Conference on Application of Natural Language to Information Systems, Springer, 2013, pp. 25–37. doi:10.1007/978-3-642-38824-8_3.

[23] A. S. Schwartz, M. A. Hearst, A simple algorithm for identifying abbreviation definitions in biomedical text, in: Biocomputing 2003, World Scientific, 2002, pp. 451–462. doi:10.1142/9789812776303_0042.

[24] N. Okazaki, S. Ananiadou, A term recognition approach to acronym recognition, in: COLING/ACL'06 Main Conference Poster Sessions, ACM, 2006, pp. 643–650.

[25] W. Zhou, V. I. Torvik, N. R. Smalheiser, ADAM: another database of abbreviations in MEDLINE, Bioinformatics 22 (22) (2006) 2813–2818. doi:10.1093/bioinformatics/btl480.

[26] J. Pustejovsky, J. Castano, B. Cochran, M. Kotecki, M. Morrell, Automatic extraction of acronym-meaning pairs from MEDLINE databases, in: MEDINFO'01, IOS Press, 2001, pp. 371–375. doi:10.3233/978-1-60750-928-8-371.

[27] S. Sohn, D. C. Comeau, W. Kim, W. J. Wilbur, Abbreviation definition identification based on automatic precision estimates, BMC bioinformatics 9 (1) (2008) 402–412. doi:10.1186/1471-2105-9-402.

[28] L. Yeganova, D. C. Comeau, W. J. Wilbur, Identifying abbreviation definitions machine learning with naturally labeled data, in: 9th International Conference on Machine Learning and Applications, IEEE, 2010, pp. 499–505. doi:10.1109/ICMLA.2010.166.

[29] A. Rajkomar, E. Loreaux, Y. Liu, J. Kemp, B. Li, M.-J. Chen, Y. Zhang, A. Mohiuddin, J. Gottweis, Deciphering clinical abbreviations with a privacy protecting machine learning system, Nature Communications 13 (1) (2022) 7456. doi:10.1038/s41467-022-35007-9.

[30] Y. Park, R. J. Byrd, Hybrid text mining for finding abbreviations and their definitions, in: L. Lee, D. Harman (Eds.), Conference on Empirical Methods in Natural Language Processing, 2001, pp. 126–133.

[31] Merriam-Webster, What is an abbreviation?, visited on 10/17/2021 (2021).
URL www.merriam-webster.com/dictionary/abbreviation

[32] L. S. Larkey, P. Ogilvie, M. A. Price, B. Tamilio, Acrophile: an automated acronym extractor and server, in: 5th ACM conference on Digital libraries, 2000, pp. 205–214. doi:10.1145/336597.336664.

[33] M. Song, P. Chang, Automatic extraction of abbreviation for emergency management websites, in: 5th International Conference on Information Systems for Crisis Response and Management (ISCRAM), 2008, pp. 93–100.

[34] Computer Hope, Computer acronyms and abbreviations, visited on 10/16/2021 (2021).
URL https://www.computerhope.com/jargon/acronyms.htm

[35] H. Hasso, K. Großer, I. Aymaz, H. Geppert, J. Jürjens, AEP-ForGTE/ILLOD: Supplemental Material v(2.1) (2022). doi:10.5281/zenodo.6884117.

[36] N. Gali, R. Mariescu-Istodor, D. Hostettler, P. Fränti, Framework for syntactic string similarity measures, Expert Systems with Applications 129 (2019) 169–185. doi:10.1016/j.eswa.2019.03.048.

[37] G. A. Miller, WordNet: a lexical database for English, Communications of the ACM 38 (11) (1995) 39–41. doi:10.1145/219717.219748.

[38] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781 (2013).

[39] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Transactions of the Association for Computational Linguistics 5 (2017) 135–146. doi:10.1162/tacl_a_00051.

[40] H. Hasso, K. Großer, I. Aymaz, H. Geppert, J. Jürjens, AEP-ForGTE/ILLOD: Supplemental Material v(1.5) (2021). doi:10.5281/zenodo.5914038.

[41] L. M. Collins, C. W. Dent, Omega: A general formulation of the rand index of cluster recovery suitable for non-disjoint solutions, Multivariate behavioral research 23 (2) (1988) 231–242. doi:10.1207/s15327906mbr2302_6.

[42] A. Jedlitschka, M. Ciolkowski, D. Pfahl, Reporting experiments in software engineering, in: Guide to Advanced Empirical Software Engineering, Springer, 2008, pp. 201–228. doi:10.1007/978-1-84800-044-5_8.

[43] Y. Jiang, H. Liu, J. Jin, L. Zhang, Automated expansion of abbreviations based on semantic relation and transfer expansion, IEEE Transactions on Software Engineering 48 (2) (2022) 519–537. doi:10.1109/TSE.2020.2995736.

[44] K. Großer, V. Riediger, J. Jürjens, Requirements document relations, Software and Systems Modeling, Theme Section Paper (2022). doi:10.1007/s10270-021-00958-y.

[45] S. Peldszus, K. Tuma, D. Strüber, J. Jürjens, R. Scandariato, Secure data-flow compliance checks between models and code based on automated mappings, in: 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS'19), 2019, pp. 23–33. doi:10.1109/MODELS.2019.00-18.

[46] K. Tuma, S. Peldszus, D. Strüber, R. Scandariato, J. Jürjens, Checking security compliance between models and code, Software and Systems Modeling (2022). doi:10.1007/s10270-022-00991-5.