

UD on Software Requirements: Application and Challenges

Naima Hassert*

Université de Montréal

naima.hassert@umontreal.ca

Pierre André Ménard*

CRIM

menardpa@crim.ca

Edith Galy

CRIM

galyed@crim.ca

Abstract

Technical documents present distinct challenges when used in natural language processing tasks such as part-of-speech tagging or syntactic parsing. This is mainly due to the nature of their content, which may differ greatly from more studied texts like news articles, encyclopedic extracts or social media entries. This work contributes an English corpus composed of software requirement texts annotated in Universal Dependencies (UD) to study the differences, challenges and issues encountered on these documents when following the UD guidelines. Different structural and linguistic phenomena are studied in the light of their impact on manual and automatic dependency annotation. To better cope with texts of this nature, some modifications and features are proposed in order to enrich the existing UD guidelines to better cover technical texts. The proposed corpus is compared to other existing corpora to show the structural complexity of the texts as well as the challenge it presents to recent processing methods. This contribution is the first software requirement corpus annotated with UD relations.

1 Introduction

Since its first release (Nivre et al., 2016), the Universal Dependencies (UD) treebank project has grown to over 200 repositories across 114 languages as of version 2.8 (Nivre et al., 2020). These treebanks target various types of documents such as news, fiction, grammar examples, spoken transcription, nonfiction, Wikipedia content, legal, religious, fiction, social media, blog, email, poetry, medical, web pages, academic, government and essays. While these types of text are often encountered, this selection leaves out one important subgenre of nonfiction: technical documents. From a natural language processing (NLP) perspective, having an annotated corpus is essential to study, evaluate and potentially train and optimize machine learning algorithms to process a specific type of text.

As a first step to study and evaluate their content, this work focus on the study of technical documents through the exploration of software requirements (SR) specifications. These types of documents often deviate from standard free-flowing text, hindering manual as well as automatic analysis of universal dependencies.

This article presents the contextual background of the study in the next section. Section 3 describes how the raw corpus was constructed, while Section 4 enumerates some phenomena that were observed in the corpus and how they were annotated. Section 5 compares the new corpus with other existing English UD annotated corpora.

2 Problem context

Technical documents, a subclass of nonfiction documents, can take several roles or forms. They can be instruction manuals, equipment maintenance procedures, documentation of schematics or plans, and so on. They might contain images, schemas, and isolated or large sections of texts, depending on their focus. Their goal might be distilled as conveying specific information in a clear, concise and unambiguous

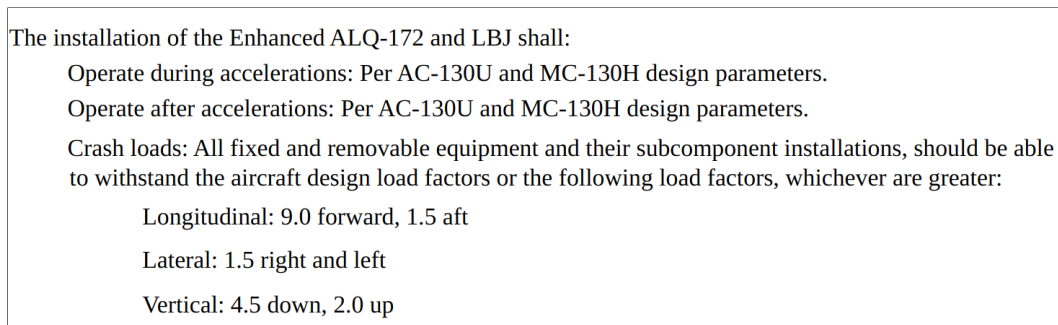
* These two authors contributed equally to this work as first authors.

way. Among this subgenre exists a specific type of technical document called software requirements specification (SRS).

Technical documents, and more specifically software requirements (SR) specifications, are written with the goal to inform the reader about a subject by giving information that is clear, measurable and unambiguous. Software requirement specifications are broken into multiple software requirements (SR). These can be analysed by software experts with the goal to unambiguously understand them and develop a software system that fulfills their needed functionalities.

Natural language processing tasks can be applied to almost any step of the software requirement life cycle, like elicitation, analysis, modeling, verification, etc (Zhao et al., 2021). Applied to software requirement specifications, dependency analysis can have multiple applications. One such use in analysis pipeline is to help perform semantic parsing (Roth and Klein, 2015) or semantic frame parsing (Wang, 2016) on SR by linking tokens to their governor, up to the root of the sentence. This step can support semantic parsing in detecting relevant parts of the sentence and attributing them specific roles such as actor, object, condition, action, etc. The result of this analysis can be used to automatically generate test cases (Ahsan et al., 2017) in order to verify software systems and improve them. Improving dependency analysis on technical documents and, more specifically, SR, can enhance the overall performance of those tasks.

Unfortunately, this is easier said than done. Several types of structural and linguistic phenomena hinder the progress of automatic dependency parsing. Manually parsing dependency relations for these texts is a difficult task for any human annotator for two reasons: the text's related technical expressions are not intuitive to understand and there are some limitations to the application of the UD guidelines. While human experts in the domain rarely have issues interpreting SR described in natural language, NLP tools trained on free-flowing texts have more difficulty in interpreting and linking sentence segments to perform dependency parsing.



The installation of the Enhanced ALQ-172 and LBJ shall:

- Operate during accelerations: Per AC-130U and MC-130H design parameters.
- Operate after accelerations: Per AC-130U and MC-130H design parameters.
- Crash loads: All fixed and removable equipment and their subcomponent installations, should be able to withstand the aircraft design load factors or the following load factors, whichever are greater:
 - Longitudinal: 9.0 forward, 1.5 aft
 - Lateral: 1.5 right and left
 - Vertical: 4.5 down, 2.0 up

Figure 1: Source document representation of a sample software requirement for a radio system.

Figure 1 shows an example of such software requirement for a radio system designed for a specific type of airplane. Briefly looking at this example, one can see multiple atypical phenomena when compared to traditional texts: partial sentences, vertical enumerations, acronyms, domain-specific named entities, etc. While this SR does not represent the majority of requirement texts in any given SRS, its composition and presentation format are common in this type of document, especially in systems that manage scientific processes or interact with physical modules. As automatic dependency parsers are usually trained on complete and well-structured sentences, they behave erratically on such texts. They fail to assign correct part-of-speech (POS) tags and to accurately detect the heads and their corresponding dependency relations. This motivates the development of annotated corpora in order to better study the represented phenomena and develop better approaches to correctly analyze them.

3 Corpus Creation Process

One of the goals for creating this SR corpus (hereafter named CTeTex for CRIM's Technical Texts corpus) was to study the performances of automatic dependency parsing on uncommon SR texts that typically cause issues with these tools.

The corpus was composed of source documents from public sources. These were taken from different websites, including one previously released dataset, which is called the PURE software requirements corpus (Ferrari et al., 2017). This last repository was used as a source to extract the majority of SR for the corpus. It contained SRS documents ranging in complexity from student projects for a multiplayer game to telescope grid communication software. The documents were produced or owned by public organizations so their licence permits free use for academic research.

Following the goal of this corpus, the requirements were chosen based on their particular attributes in order to present various features of software requirement documents and, more broadly, technical documentation. Documents from the corpus were scanned manually in order to differentiate SR from non-requirements segments (section’s introduction, generic explanations, etc.) and were sampled based on the expected challenge they offered. While a random selection would have been more representative of the given corpus, this selection process gives a better view of the problematic issues presented by these types of documents.

The Inception platform (Klie et al., 2018) was used to perform dependency annotation. As it performs default tokenization on texts, tokens were corrected manually. Basic UD dependency relations were added to the tokens without the enhanced version. The resulting corpus¹ contains 196 SR coming from 24 source documents with 9,273 tokens distributed in 276 sentences. An annotator syntactically analyzed each document to produce token segmentation, part-of-speech tags and labelled dependency relations. The annotator was a student mastering in linguistics with prior knowledge of dependency grammar and was trained with a senior linguist on an alternate proprietary technical corpus to ensure the quality of the annotations. The initial Kappa inter-annotator agreement on the first segment of the training corpus was 0,69, but grew to near-perfect match on non-problematic cases through iterative consulting with the senior linguist. This ensured that the UD guidelines were understood and homogeneously applied by the annotator and the senior linguist.

For the CTeText corpus, the annotator consulted with a software engineer in order to clarify technical terms and expressions that were specific to the software or engineering domain, a type of system or a single system. This helped clarify some ambiguity on both part-of-speech tags and dependency relations. The annotations were then revised by a group of three experts (including the annotator) to discuss unresolved cases in order to solve them. The annotation process took approximately 180 hours, considering only the main annotator’s time.

4 Applying UD Guidelines on Software Requirements

This section describes various structural and linguistic phenomena that are often found in SR texts, the issues in applying the guidelines and how they were annotated in the proposed corpus. Table 1 shows the quantity of examples found in the corpus for each of the subsections. Note that some text segments, like a scientific notation using abbreviations and specialized vocabulary, can be counted in multiple categories.

Phenomena	Occurrences	Number of tokens
Scientific and mathematical notations	22	68
Abbreviations and acronyms	579	579
Lists and enumerations	41	2 736
Specialized vocabulary	503	1 180

Table 1: Estimation of occurrences and their total number of tokens for each phenomenon in CTeText.

For each issue detected during the annotation phase, an in-depth analysis was done in order to find the most adequate proposition. Each of the three experts revisited the relevant UD definitions of all the possible alternatives to solve the issue. They then looked at the other English corpora (referred in Section 5) with the Grew-match² online search tool to check if there was similar cases that could

¹The corpus will be published on the Universal Dependencies repository under CTeText name.

²<http://match.grew.fr>

shed some light on the targeted challenge and if they could apply to the context of CTeTeX corpus. Specialized online discussions on universal dependencies were also consulted when topics aligned with one or multiple possible solutions. Individual findings were then discussed together in order to review the possible options and evaluate which one was the best to express the syntactic aspect in the specialized context of technical texts.

4.1 Scientific and Mathematical Notations

Mathematical formulas and scientific expressions are often found in SR texts in order to inform the reader on the valid response the described system should provide. However, the UD documentation does not address which dependency relations are appropriate for those specific types of construction. One of the only referenced cases related to those expressions is the specification that mathematical operators should be designated as a *SYM* POS tag. The following sections provide details about the annotation choices made for the CTeTeX corpus as a well as a proposition to enhance the UD guidelines.

4.1.1 Formulas

Formulas use variables, coefficients and operators to clearly communicate a scientific or mathematical process that should be implemented by a system, as shown in the underlying sentence in Figure 2. In an attempt to determine which dependency relations were adequate in these cases, the verbalization of those mathematical formulas and expressions was first considered. What could one say if one wanted to express the formula in words? It was then proposed that the symbol “=” could be considered as the equivalent of the verb “equals” or “is”, as the head of the expression, the symbol “+” as the coordinating conjunction “plus” or “and”, etc.

However, the verbalization of those expressions is far from straightforward. For example, the expression $O(n^2)$. If the annotator does not have a background in mathematics or computer science, it could be difficult to come up with a valid verbalization that would translate to “a big O of n squared”³.

The solution proposed is thus to acknowledge that mathematics is a formal language and, by nature, does not obey the syntactic rules of natural languages like English. These are in fact two different languages. When the two of them are found in the same text, it is a case of code-switching. Trying to analyze both of them in the same way could result in misleading annotations.

Fortunately, UD already has a way to deal with foreign languages: the relation *flat:foreign*. It is suggested that the head of the mathematical expression should be what is considered to be its first token, for simplicity reasons. In accordance with UD guidelines, this head should be the parent of all the following tokens constituting the mathematical expression. An example of the resulting tree can be seen in Figure 2.

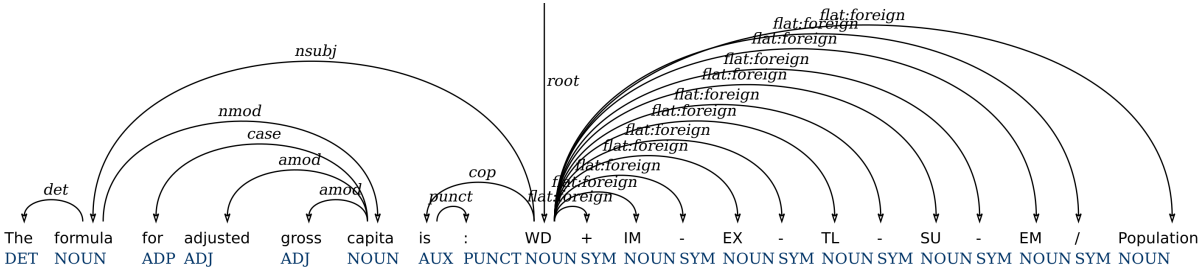


Figure 2: Annotated mathematical expression.

This solution has several advantages; it does not require the annotator to verbalize expressions that are not intended to be expressed in words, it is constant and easily applicable to any specific case, it uses a dependency relation that already exists in the UD documentation, and it differentiates them from text written in natural language in the corpus.

However, the presence of mathematical operators does not necessarily indicate the presence of a mathematical formula or expression. Sometimes, the equal sign is found alone and is used to indicate the

³See https://en.wikipedia.org/wiki/Big_O_notation

meaning of a word or a group of words. In that case, the verbalization approach is more appropriate. As it only implicates a single operator in this case, the proposed solution is to treat the equals sign as a verb. The subject is the word that precedes it and the object is the word that follows it. This is illustrated in Figure 3. Note the error in the sentence : 'groundwater' should be one word (see 'goeswith' between 'ground' and 'water'). That is why 'ground' is coordinated with 'water' differently than 'surface'.

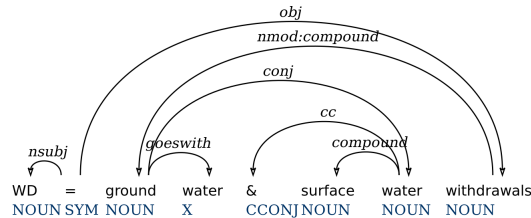


Figure 3: Example of a standalone equal operator.

4.1.2 Variables

Mathematical formulas suggest that variables and placeholders for undefined values will be present in SR texts. For example, in “*less than t minutes*” or “*for n nodes*”. Regarding the part-of-speech tag that should be attributed to variables, the UD documentation does not provide an easy solution. It mentions: “The universal POS tags should capture regular, prevailing syntactic behavior, as well as morphological characteristics when available, and should not reflect sentence-specific exceptional behavior.”

But what is the “prevailing syntactic behavior” of a letter? In the online Merriam-Webster dictionary, letters are considered as nouns when they designate the letter in the alphabet or when used as an abbreviation for a noun that begins with that letter. Nothing is mentioned for when it is a name of a variable. Two choices were thus considered: the UPOS *X*, since no official answer yet exists, or *NUM*, since it is clear that it is what the variable expresses. UD guidelines specify that the UPOS *X* should be used sparingly, and only when there is no other possibilities. The second solution was thus the one adopted. The dependencies are then defined as the sentence dictates, as if a number was replacing the variable. While there is no case in the proposed corpus, a variable with a different type of implied value (categorical, boolean, string, etc) would be tagged as if a specific value was used, likely with *NOUN* (i.e. *The system will send the s string*).

Is it important to note that this proposition excludes named system variables like in “display the content of the XYZ field” in which the name of the variable is a recognized concept of a system and is named to differentiate it from other similar concepts, thus behaving like a noun and were tagged with *PROPN*. This is different from an unnamed variable like “display the top n results” which behaves syntactically like its underlying numerical value.

4.2 Abbreviations and Acronyms

Software requirements usually contain high number of acronyms, mainly from computer science but also from the described system’s application domain. Specifically, acronyms of proper and common nouns, abbreviations of short expressions and Latin abbreviations have been found in the corpus. But information on acronyms and abbreviations is very scarce in UD documentation. This section presents an analysis for these types of constructions. For easier reference, the relevant expressions in the section are provided with the *Abbr=Yes* feature in the treebank.

4.2.1 Acronyms

As mentioned in section 2, software requirements are written with the goal of informing the reader about the software in clear and precise language. That is why this type of document refers to various components of software that often have long, specific and repetitive names. It is then normal that in order to reduce the length of the text and to facilitate its reading, those names are shortened in the form of acronyms.

These types of acronyms are a challenge for an annotator for two reasons. First, the UD documentation does not give clear guidelines for the appropriate POS tag to give to acronyms. It only mentions one case: “Acronyms of proper nouns, such as UN and NATO, should be tagged PROPN.” Nothing is said about acronyms of common nouns or acronyms of nominals of which the head is a common noun, like “ID” (“Identifier”) or “ETA” (“Estimated Time of Arrival”). It was then decided that acronyms of common nouns should be tagged *NOUN*.

There is a second challenge for the annotator: to find, in the document itself or elsewhere if it is not specified in it, the long form of the acronym. The alternative use of short and long forms of acronyms, without explicit association, is common in organization’s internal documents (Ménard and Ratté, 2010). It is crucial in order to understand the general meaning of the requirement, but also for deciding if the appropriate POS tag is *PROP*N or *NOUN*.

It should be noted that the difference between proper and common nouns can be very subtle, and it was not the objective of this paper to address this problem. However, it was decided that within the corpus, proper nouns are nouns that designate a specific entity that can be distinguished in a group of similar entities. The following requirement can be taken as an example.

The TCS will provide the hardware and software necessary to allow the operator to conduct the following major functions 1) mission planning, 2) mission control and monitoring, 3) payload product management, 4) targeting, and 5) C4I system interface.

“TCS” here means “Tactical Control System”. It designates a specific system that can be distinguished in a group of similar systems and that the entire technical document is meant to describe. It is then annotated as a proper noun. However, “C4I” is a type of system, not a specific system: it is then a common noun.

4.2.2 Abbreviations of Short Expressions

When short expressions were abbreviated, like “TBD” (“To Be Determined”) or “TBC” (“To Be Confirmed”), it was decided to give them the POS tag of the head of the expression (*VERB* in the mentioned examples). This is because they could easily have been written in their long form and the normal syntactic behavior of the sentence would have been preserved. The dependency relation is thus the one that would have been appropriate if the expression was complete, as illustrated in Figure 4. Note that while “TBC” is a parataxis, it is considered to qualify the noun “function” and was analyzed as a long form that would be inserted after the head noun “function”. It is thus linked as *acl*.

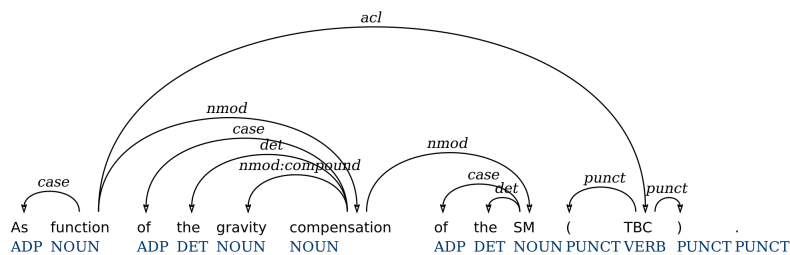


Figure 4: Annotated sentence excerpt of an abbreviated short expression.

However, those abbreviations can sometimes be used as a placeholder for a number to indicate that a value has not yet officially decided. In this case, if the expression was presented in its long form, the sentence would be unnatural and even faulty. For the same reason variables’ names were tagged as *NUM*, it was decided that in these cases, the POS tag should be *NUM*. This option was chosen so the POS tag would reflect the fact that it is very clear that “TBD” stands for a number and has the same role. Figure 5 illustrates this analysis.

4.3 Lists and Enumerations

One of the abundant yet problematic syntactic constructions in technical documents from the NLP point-of-view is vertical lists. Although UD guidelines specify what is the correct relation to apply within

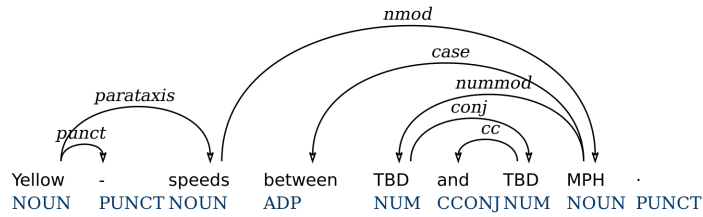


Figure 5: Annotated sentence excerpt of an abbreviation used as an undefined number.

In the Normal Operations Mode the TCS shall support the following functions: [SSS037]

1. Mission Planning
2. Mission Control and Monitoring
3. Payload Product Management
4. Target Coordinate Development
5. C4I Systems Interface

(a) Full introduction.

When the system receives a Flight Plan message or an Airline Flight Create message to create a new flight that has:

- a. Same flight ID as an existing flight, and
- b. Same destination (but with different origin) as that existing flight, And that existing flight has:
- c. A departure time within 10 hours prior to the new flight's departure time, and
- d. A status of "cancelled", i.e. 'diversion cancelled', the system shall identify the new flight to be an 'auto' Diversion Recovery flight.

(b) Fragmented introduction.

Figure 6: Two types of introduction sentence for list.

the elements of a list (*list*), which is only used to connect list items that do not appear in a standard syntactic construction, such as coordination, they do not specify the relation that should be used to link the introductive part of the sentence before the colons and the first element of the list.

In addition, the CoNLL-U file format does not allow line skips in sentences, nor does it allow for multiple lines of text definition in the header. This prohibits the lossless representation of sentences spanning multiple lines, as in the case of vertical lists.

4.3.1 Relation for List Introduction

Two types of syntactic structures were seen in requirements containing lists. One type uses a complete sentence as an introduction to the list, containing a verb, its subject and a complement as shown in Figure 6a. In the other type, the clause that precedes the colon is an incomplete sentence, ending with a transitive verb or any other word that needs an argument (Figure 6b).

The proposed solution for the first type of construction is to use the relation *parataxis* between the head of the clause, to which the list is linked to, and the first element of the list. For the second type of construction, however, as the part of the sentence before the colons is incomplete, the *parataxis* relation is not the best relation to use. The proposed approach (Figure 7) defines the relation between the verb ("has" in the example below) and its direct object ("ID") as the habitual *obj*. More broadly, the relation to use is the one that would have been obvious without the colons. The resulting tree is represented below.

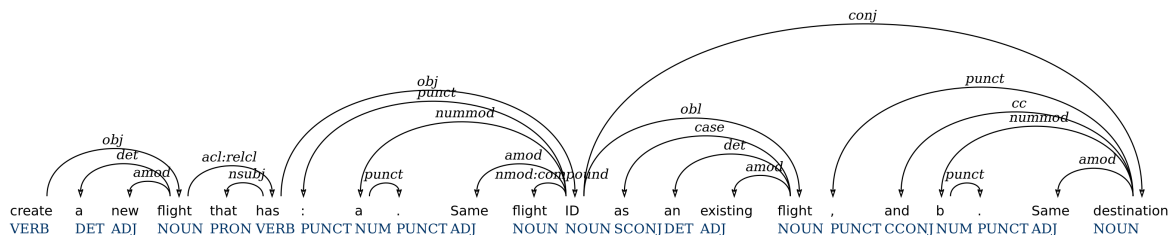


Figure 7: Example of an annotated list introduction .

4.3.2 Multilevel Lists

Multilevel lists occurred several times in the CTeX corpus. This type of construction seems specific to technical documents. Indeed, there is no mention of them in the UD documentation and no examples were found in the other English UD corpora. However, it is easy to represent this syntactic structure with the UD relation *list*. To differentiate the different levels of lists from each other, the annotator simply has to consider them as different lists, with different heads. In the Figure 8, for example, the first list is composed of the phrases “Target CCTV” and “Device Control”, while the second list, or sublist, is composed of the words “Pan”, “Tilt” and “Zoom”. The first list then begins by “CCTV”, linked with its head “provide” with the relation *parataxis* (not with the noun “information”, as explained in Section 4.3.1). The second list begins with “Pan”, linked with its head “control” (which is the last element of the first list) with the relation *nmod*.

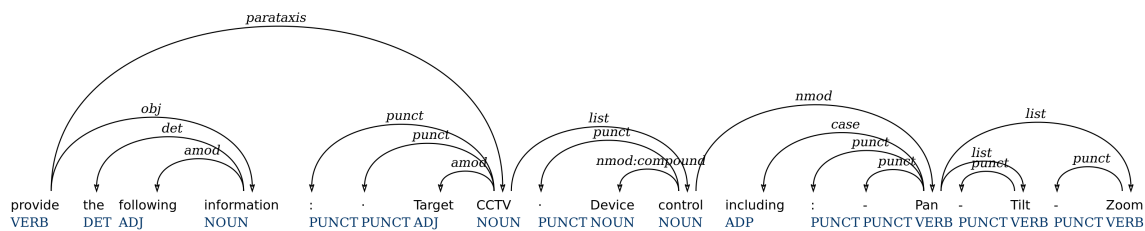


Figure 8: Example of an annotated multilevel list.

4.4 Specialized Vocabulary

Specialized vocabulary is perhaps the most obvious challenge when annotating technical documents. Annotators or readers who are not experts in the field may have trouble understanding words and multiword expressions in the text, making the annotation process longer and more complex.

4.4.1 Meta Identifiers

Some documents refer to specific SR using a unique identifier throughout the document, as shown in the example below. This identifier is sometimes situated inside the requirement (in those cases, it is usually placed after the main tensed verb of the requirement), and sometimes outside it.

The scheduler shall [SRS181] set the 50 Hz interval timer to a count down value so as to cause the next minor frame interrupt at 20 msec from the previous interrupt congruently in all operational FCPs.

Here, the meta identifier *SRS181* is used to name the entire requirement. Elsewhere in the document, this requirement can be referred using this identifier. It would thus be natural to give it the POS tag *PROPN*.

The UD relation that seemingly describes this instance better is the *appos* relation. Even though the referent is not usually a noun (but rather a full sentence, thus a verb parent) as required by this relation’s definition. It also does not immediately follow its parent, as also defined in the guidelines. Nonetheless, it is proposed to extend the definition of the *appos* relation to include constructions with meta identifiers. This is because it is the only relation that really captures the function of those identifiers, which is to name the requirement. An example of the suggested analysis is represented in Figure 9.

4.4.2 Nominal Modifiers vs Compounds

Using specialized technical vocabulary makes it more difficult to differentiate between the dependency relations *nmod* (nominal modifier) and *compound*. It is probably the most influential UD guideline regarding manual annotation of technical documents, because of the wide number of cases where a decision has to be made.

At the moment, UD guidelines differentiating between compositionality and nominal modifiers are unclear: for example, in the UD documentation, “phone book” is treated as a compound (in which the

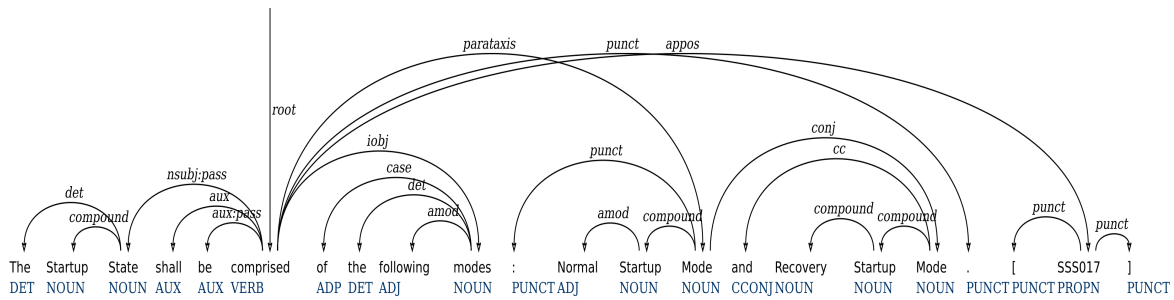


Figure 9: Example for the meta identifier identification

meaning is not compositional), as well as “ice cream flavors”, of which the meaning is compositional (at least between “flavors” and “ice cream”).

It is therefore proposed to follow one part of Sylvain Kahane’s recommendation in this Github discussion ⁴. As he suggests annotating all the Noun Noun combination with a new *nmod:compound* sub-relation, we propose to use it for word group whose meaning is compositional and which could be paraphrased like (like “the dog tail” - “the tail of the dog”). This sub-relation does not yet exist, but it has the advantage of identifying borderline cases, and could easily be modified in post-processing, if a different approach is chosen.

In summary, if the nominal expression is idiomatic (non-compositional), then it is a *compound*. If it is compositional but does not possess any sort of case marking, then it is a *nmod:compound*. Finally, if it is compositional and it possesses a case marking, then it is a *nmod*.

4.4.3 Meta Qualifiers

As shown in Section 4.4.1, some requirements are accompanied by meta identifiers that are used to name the entire requirement. Similarly, some requirements are accompanied by qualifiers used to specify the status of the requirement. In the CTeTex corpus, such qualifiers indicated if the requirements were optional (“O”), mandatory (“M”), or were used as an element of information (“I”) (the meaning of those letters were found directly in the document). In the following example, the requirement is qualified as mandatory.

The network shall terminate the ongoing VCS/VBS call if it receives the 3-digit sequence “***” transmitted via DTMF signals. (M)

Those types of constructions were not found in the UD documentation. It is rather unusual to witness, in other types of documents, an element that qualifies an entire sentence instead of another word. They are the equivalent, in other technical documents, or prefixing the sentence with “It is mandatory that the systems ...”, which specifies the modality of the requirement, much like an adverbial modifier. The current case play the same semantic role, but is outside of the grammatical structure of the modified sentence. While *advmod* was considered, the *parataxis* is used to link them to the root of the requirement. The relation is applied to the “M” and “O” qualifiers which are considered as having the *ADJ* POS tag, or *NOUN* for the “I” qualifier. The resulting tree is illustrated in Figure 10.

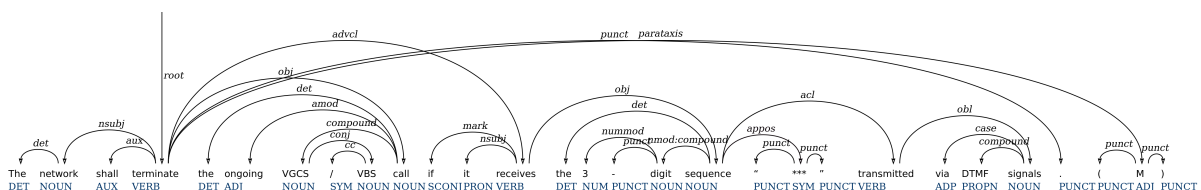


Figure 10: Annotated example of a meta qualifiers in a software requirement.

⁴<https://github.com/UniversalDependencies/docs/issues/761>

In cases where requirements are constituted of multiple sentences, it is suggested to link those indications to the head of the last sentence rather than the first one. This is to avoid bias in the statistics of sentence size and distance between tokens.

4.5 Issues with CoNLL-U Format

Other than the challenge of applying the guidelines to the CTeTex corpus, using the current CoNLL-U file format to express UD annotations is also problematic. One is that this format is lossy for multiline sentences, as there is no mechanism to express a line change within a sentence, so the resulting sentence cannot be rebuilt correctly. Another is that splitting multiline sentences into separate entries in the file will force the loss of list references as there is no cross-sentence dependency indicator in CoNLL-U format.

One way to solve these issues would be to allow intersentence parent reference with a [sentence id]-[token id] structure to avoid multiline sentences in the current format. While that would help, it would disrupt the semantics of the format by enabling the splitting of a sentence while also changing the format of the parent id. While it is feasible, it has much impact on existing resources and codebase in CoNLL-U format. A simpler solution would be to add a "LineAfter=Yes" attribute in the *MISC* column to encode the line skip characters, so that it would be possible to reconstruct the exact format of the sentence. The latter option was retained to encode the CTeTex corpus as it has the least impact on the file format and it helps solve the two issues.

5 Corpora Comparison

Following the application of the guidelines, there is a need to validate if these syntactic constructs really differ from the other typical UD corpora and if they affect automatic processing tools. To that end, CTeTex is compared with other English corpora in order to view the differences in performances when a dependency parser is applied. The EWT (Silveira et al., 2014), GUM (Zeldes, 2017), GUMReddit (Behzad and Zeldes, 2020), ParTuT (Sanguinetti and Bosco, 2015), PUD (McDonald et al., 2013), LinES (Ahrenberg, 2015), Pronouns (Munro, 2021) and ESL (Berzak et al., 2016) corpora (version 2.8 of the UD dataset) serve as a basis of comparison. Some of these texts targets a specific linguistic phenomenon (Pronouns), are manually or semi-manually annotated (EWT, GUM, etc).

The left part of Table 2 shows the average sentence's length, height (or depth), arity and mean dependency distance or MDD (Jiang and Liu, 2015) for each of the eight corpora, followed by the average measure over these same corpora. The CTeTex measures are then shown, with their differences (Δ) with the actual corpora average.

The average sentence length of CTeTex is almost 50% longer and around 10% deeper than the second-highest measures from ParTUT corpus. This is expected as vertical lists often contain multiple elements that directly influence the length of the overall sentence. While arity (number of children for a node) is close to the ESL corpus, it is almost 20% higher than the average corpus. The complexity and length of sentence also influence the MDD of CTeTex which is the highest among all corpora. This indicates that tokens are less related to close neighbours that in other corpora, but are linked to parents that are often found at a greater distance in the sentence.

This might have a negative impact on automatic tools if they use a smaller contextual window to search for parent tokens. It also impacts the complexity of the annotation process, as the cognitive load of understanding complex sentence can hinder the speed of analysis.

To evaluate the influence of the nature of the texts of the proposed corpus, the nine English UD corpora were automatically annotated using Stanza v1.2.3 (Qi et al., 2020) dependency parser. Other dependency parsing tools (like Spacy (Honnibal and Montani, 2017) and UDPipe 2 (Straka, 2018)) were also tested, but produced worst overall performance on the CTeTex corpus as well as the other UD corpora. Thus only Stanza's results are presented for brevity. It should be noted that most universal dependency parsers are trained to use some version of the English UD dataset, as few other resources are publicly available. This is a methodological issue for the referred eight UD corpora as training data is usually not used for evaluation. But the hypothesis was that if CTeTex was similar to existing UD treebanks, the difference

(Δ) between the average and CTeTex scores (for the automatic annotation columns) would have been relatively small. The delta values thus emphasize the remote nature of the content of CTeTex compared to existing annotated texts.

Corpus	Sentence (avg)				Automatic annotation			
	Length	Height	Arity	MDD	UPOS	UAS	LAS	CLAS
EWT	15.33	3.32	4.66	3.38	0.9685	0.9345	0.9173	0.9003
GUM	18.17	3.72	4.83	3.39	0.9543	0.8963	0.8744	0.8570
LinES	17.97	3.75	5.08	3.30	0.9246	0.8237	0.7828	0.7537
ParTUT	23.75	4.71	5.52	3.48	0.9156	0.8602	0.8038	0.7565
Pronouns	5.98	1.81	3.44	1.76	0.9599	0.8519	0.8171	0.8397
PUD	21.18	4.31	5.76	3.34	0.9586	0.8882	0.8626	0.8454
GUMReddit	18.20	3.77	5.21	3.41	0.9439	0.8585	0.8278	0.8114
ESL	19.06	3.97	5.85	3.31	0.9368	0.8999	0.8643	0.8464
Average	17.71	3.67	5.04	3.17	0.9452	0.8766	0.8437	0.8263
CTeTex	33.60	5.17	6.01	3.98	0.8699	0.7739	0.6879	0.5949
Δ	15.89	1.5	0.97	0.81	0.0753	0.1027	0.1558	0.2314

Table 2: Overview of English UD corpora compared to CTeTex. (highest values in bold)

The right section of Table 2 shows universal part-of-speech (UPOS), unlabelled association score (UAS), labelled association score (LAS) as well as the labeled association score for content words (CLAS) like nouns, verbs and adjectives. Best overall scores were obtained on the EWT corpus. One explanation might be the larger size of this resource compared to other corpora when used as training data.

While the existing corpora offer a good score for the UPOS tag, the performance on CTeTex is 7.52% lower. The reasons for such a low score might be explained by some of the decisions in Section 4 (variable as *NUM*, etc.) but also by the large number of acronyms and complex domain specific terminology. The scores continue to degrade with the addition of dependency relations (UAS), their labels (LAS) and the specific study of content words (CLAS), dropping by 23.14% from the average corpora on this last score. This goes to show that the complexity of SR texts and their underlying phenomena hinders current dependency parsers. Using information analyzed by these tools in downstream processing tasks lowers the chance of a usable outcome. The CTeTex corpus is thus a relevant contribution to improve the adaptability and stability of dependency parsers when processing technical document such as software requirements specification.

6 Conclusion

This contribution is the first English software requirements corpus annotated with Universal Dependencies part-of-speech and labelled dependency relations. The comparison to other existing corpora in English shows specificity of the CTeTex corpus as well as the challenge of processing such texts with automatic dependency annotation tools. It offers the possibility to evaluate and experiment on this type of document to improve both the UD guidelines and the automatic annotation process.

Future work on this corpus includes studying the addition of enhanced UD relations as a way to better express needed links. This would permit a better extraction of higher-level information from the requirements. Future plans also include using CTeTex to train and evaluate neural dependency parsing algorithms to improve their performance on this type of technical documents.

References

Lars Ahrenberg. 2015. Converting an English-Swedish parallel treebank to Universal Dependencies. In *Proceedings of the Third International Conference on Dependency Linguistics (Depling 2015)*, pages 10–19, Uppsala, Sweden, August. Uppsala University, Uppsala, Sweden.

- Imran Ahsan, Wasi Haider Butt, Mudassar Adeel Ahmed, and Muhammad Waseem Anwar. 2017. A comprehensive investigation of natural language processing techniques and tools to generate automated test cases. In *Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing, ICC '17*, New York, NY, USA. Association for Computing Machinery.
- Shabnam Behzad and Amir Zeldes. 2020. A cross-genre ensemble approach to robust Reddit part of speech tagging. In *Proceedings of the 12th Web as Corpus Workshop (WAC-XII)*.
- Yevgeni Berzak, Jessica Kenney, Carolyn Spadine, Jing Xian Wang, Lucia Lam, Keiko Sophie Mori, Sebastian Garza, and Boris Katz. 2016. Universal dependencies for learner english. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 737–746. Association for Computational Linguistics.
- Alessio Ferrari, Giorgio Ortonzo Spagnolo, and Stefania Gnesi. 2017. Pure: A dataset of public requirements documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 502–505.
- Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.
- Jingyang Jiang and Haitao Liu. 2015. The effects of sentence length on dependency distance, dependency direction and the implications—based on a parallel english–chinese dependency treebank. *Language Sciences*, 50:93–104.
- Jan-Christoph Klie, Michael Bugert, Beto Boullosa, Richard Eckart de Castilho, and Iryna Gurevych. 2018. The inception platform: Machine-assisted and knowledge-oriented interactive annotation. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 5–9. Association for Computational Linguistics, June.
- Ryan McDonald, Joakim Nivre, Yvonne Quirnbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. Universal Dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria, August. Association for Computational Linguistics.
- P. Ménard and S. Ratté. 2010. Classifier-based acronym extraction for business documents. *Knowledge and Information Systems*, 29:305–334.
- Robert Munro. 2021. *Human-in-the-Loop Machine Learning*. Manning.
- Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia, May. European Language Resources Association (ELRA).
- Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: An evergrowing multilingual treebank collection. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4034–4043, Marseille, France, May. European Language Resources Association.
- Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.
- Michael Roth and Ewan Klein. 2015. Parsing software requirements with an ontology-based semantic role labeler. In *Proceedings of the 1st Workshop on Language and Ontologies*, London, UK, April. Association for Computational Linguistics.
- M. Sanguinetti and C. Bosco. 2015. Parttut: The turin university parallel treebank. In *Italian Natural Language Processing within the PARLI Project*.
- Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Christopher D. Manning. 2014. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.
- Milan Straka. 2018. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207, Brussels, Belgium, October. Association for Computational Linguistics.

- Yinglin Wang. 2016. Automatic semantic analysis of software requirements through machine learning and ontology approach. 21:692–701, Dec.
- Amir Zeldes. 2017. The GUM corpus: Creating multilayer resources in the classroom. *Language Resources and Evaluation*, 51(3):581–612.
- Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J. Letsholo, Muideen A. Ajagbe, Erol-Valeriu Chioasca, and Riza T. Batista-Navarro. 2021. Natural language processing for requirements engineering: A systematic mapping study. *ACM Comput. Surv.*, 54(3), apr.