

Comparative Evaluation of NLP Approaches for Requirements Formalisation

Shekoufeh Kolahdouz Rahimi¹^a, Kevin Lano²^b, Sobhan Yassipour Tehrani³^c,
Chenghua Lin⁴^d, Yiqi Liu⁴^e and Muhammad Aminu Umar²^f

¹*School of Arts, University of Roehampton, London, U.K.*

²*Department of Informatics, King's College London, London, U.K.*

³*Department of Computer Science, University College London, U.K.*

⁴*Department of Computer Science, University of Sheffield, U.K.*

Keywords: Requirements Formalisation, Model-Driven Engineering, NLP.

Abstract: Many approaches have been proposed for the automated formalisation of software requirements from semi-formal or informal requirements documents. However this research field lacks established case studies upon which different approaches can be compared, and there is also a lack of accepted criteria for comparing the results of formalisation approaches. As a consequence, it is difficult to determine which approaches are more appropriate for different kinds of formalisation task. In this paper we define benchmark case studies and a framework for comparative evaluation of requirements formalisation approaches, thus contributing to improving the rigour of this research field. We apply the approach to compare four example requirements formalisation methods.

1 INTRODUCTION

Automated requirements formalisation (RF) has significant potential as a means of reducing software development costs, accelerating development processes, and increasing the rigour of requirements engineering processes. Typically, RF involves producing a software model in a language such as UML, or in a domain-specific language (DSL), from natural language requirements documents in text format. The RF process or result can also provide useful analysis information about the requirements statement, i.e., to detect duplicated or invalid requirements.

Many approaches have been proposed for the formalisation of software requirements, typically involving some form of natural language processing (NLP) or machine learning (ML) (Otter et al., 2023). However, the research field lacks widely-recognised benchmark

case studies to support comparative evaluation of different approaches on the same requirements cases, and the main evaluation technique, based on estimating precision/recall and F-measure accuracy, is subjective.


F-measure gives the degree to which a proposed formalised model correctly expresses the model elements implied by the source text, and has the definition


$$F = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$


$$\text{recall} = \frac{\text{correctly identified elements}}{\text{total identified}}$$


$$\text{precision} = \frac{\text{correctly identified elements}}{\text{total correct elements}}$$


The judgement as to the correctness of the model elements has a subjective aspect, for example, it may be based on agreement with a 'gold standard' model produced by a human expert. However different modellers may produce significantly different gold standard models. Our view is that evaluation of RF approaches should use objective measures where possible, and also take into account the software engineering context of use of formalised models. In general, software


^a <https://orcid.org/0000-0002-0566-5429>

^b <https://orcid.org/0000-0002-9706-1410>

^c <https://orcid.org/0000-0003-4417-0477>

^d <https://orcid.org/0000-0003-3454-2468>

^e <https://orcid.org/0000-0002-8070-5056>

^f <https://orcid.org/0000-0001-9433-2409>

developers should be able to effectively use the formalised models for further development stages, and should be able to relate the models to the original requirements statement. Thus the quality and internal consistency of the formalised model is important (e.g., there should not be duplicated class or use case names in formalised models, and names should adhere to name style conventions for the respective model kinds). There should be a high degree of traceability between the formalised requirements and the source text.

To effectively compare different RF approaches, objective measures of accuracy are needed, which are aligned to the SE context of use of the formalised model. Thus we propose the evaluation model of Figure 1, which involves three aspects: (i) an objective measure of similarity between the formalised model and a rigorously produced ‘gold standard’ reference model; (ii) a measure of completeness of the formalised model wrt the source requirements document; (iii) a measure of internal quality of the model.

The reference model could be produced by agreement between two or more human experts, with an independent review by a further expert, to improve its appropriateness as a correctness standard.

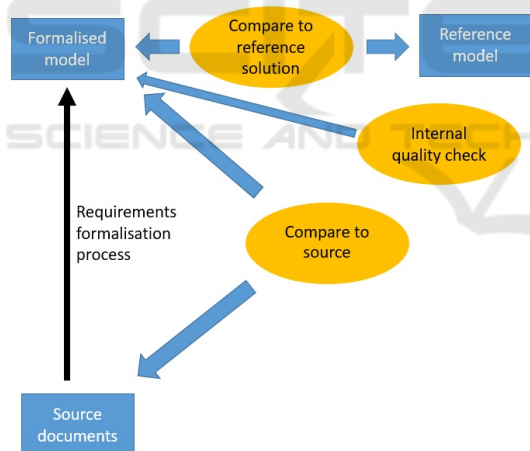


Figure 1: Evaluations of requirements formalisation.

1.1 Natural Language Processing (NLP)

NLP is a collection of techniques for the processing of natural language text, including part-of-speech (POS) tagging/classification, tokenisation and segmentation, lemmatisation, parsing, chunking, dependency analysis and reference correlation. NLP tools include Stanford NLP (Stanford University, 2020), Apache OpenNLP (Apache, 2021), iOS NLP Framework, Python’s NLTK, and WordNet (Princeton University, 2021).

The standard parts of speech include (Santorini, 1990): Determiners – tagged as *DT*, e.g., “a”, “the”; Nouns – *NN* for singular nouns and *NNS* for plural; Proper nouns – *NNP* and *NNPS*; Adjectives – *JJ* for general adjectives, *JJR* for relative adjectives, *JJS* for superlatives; Modal verbs – *MD* such as “should”, “must”; Verbs – *VB* for the base form of a verb, *VBP* for present tense except 3rd person singular, *VBZ* for present tense 3rd person singular, *VBG* for gerunds, *VBD* for past tense; Adverbs – *RB*; Prepositions/subordinating conjunctions – *IN*.

For specialised purposes, additional parts of speech may be defined, as in (Xu et al., 2020). NLP has been a key element of automated RE approaches (Otter et al., 2023). However, the trained models available for POS-tagging and parsing with the existing NLP tools are usually oriented towards general English text, which differs significantly from the subset of English typically used in software requirements statements. The existing POS-tagger models therefore sometimes misclassify words in requirements statements (e.g., “stores” may be misclassified as a noun, even when used as a verb, and “existing” used as an adjective misclassified as a gerund).

1.2 Machine Learning (ML)

Machine learning covers a wide range of techniques by which knowledge about patterns and relationships between data is gained and represented as implicit or explicit rules in a software system. ML can be used for classification, translation or prediction. In particular, ML is used to create the part-of-speech and other models used in NLP tools. ML techniques include K-nearest neighbours (KNN), decision trees, inductive logic programming (ILP) and neural nets. A key distinction can be made between techniques such as decision trees and ILP where explicit rules are learnt from data, and techniques such as neural nets where the learned knowledge is in an implicit form (consisting of the weights of connections in the trained network). In recent years there have been substantial advances in neural networks (recurrent neural networks or RNNs) which possess a ‘memory’ of a sequence of inputs, enabling them to perform prediction tasks and process data (such as natural language texts or programs) that consist of a connected sequence of elements (sentences) (Kolahdouz-Rahimi et al., 2023).

The increasing power of ML approaches based on large language models (LLMs) such as BERT (Guo et al., 2021), Codex (Chen et al., 2021) and GPT (Zhao et al., 2023) has already led to innovative software assistants such as Copilot (GitHub.com, 2022) and program translators such as CodeT5. The appropriate

pre-training and fine-tuning of LLMs to support software engineering tasks is an area of active research. There have been few works on using LLMs for requirements formalisation and existing LLM-based tools are limited in their capabilities in this area (Camara et al., 2023).

Toolsets for ML include Google MLKit, TensorFlow, Keras, ScikitLearn and Theano.

2 COMPARED APPROACHES

In this study we illustrate our proposed evaluation framework by comparing four alternative approaches for formalising behavioural system requirements expressed in unstructured English text or as semi-structured user stories.

The approaches are:

Hamza and Hammad (Hamza and Hammad, 2019): based on segmentation, POS-tagging, chunking, grammar patterns.

Elallaoui, Nafil and Touahni (Elallaoui et al., 2018): based on POS-tagging

AgileUML (Lano et al., 2021): based on POS-tagging, chunking, semantic analysis and word-similarity matching

Simple heuristic: As a baseline for comparison, a simple heuristic approach based on POS-tagging and chunking is defined and evaluated.

User stories are widely-used in agile methods to express functional requirements. They have the semi-structured format

As a/an [actor], I [wish/want/...] to [action], [purpose]

where the stakeholder requiring the functionality is identified in the first part, then the action in the second part, and an optional purpose is described in the third part.

For example:

As a doctor, I wish to view the patient's EHR

2.1 Hamza and Hammad Approach (Hamza and Hammad, 2019)

The approach starts from a textual specification of requirements in English text, this is spell-checked to eliminate erroneous text, then segmented into sentences. POS-tagging is applied, and this information is used to chunk the text into sequences of closely-associated words. For example the sequence of adjectives associated with a noun are grouped with it: this is the chunk pattern JJ^*NN and related chunk patterns.

Stemming is used to identify the root form of words in the text. Grammar knowledge patterns (GKP) are used to recognise expected sentence structures, and provide a corrective to semantic errors arising from incorrect POS-tagging.

To identify actors and actions of a use case, different rules are applied to handle variation in the way that these can be expressed in unstructured text.

The approach is evaluated on four case studies of requirements statements, of small size (each case has between 11 to 23 functional requirements). It is not clear if these are real systems or artificial examples. Precision and recall are evaluated, however it is unclear how the correctness of the formalisation is determined, ie., how the correct reference model was constructed. They find an average recall of 69% and precision of 72%, which indicates that the approach tends to produce both incorrect formalisations, and fails to produce correct formalisations. The reasons for these errors are mainly due to linguistic variability and ambiguity/incompleteness in the requirements statements. Another factor is that the requirements statements also express constraints, such as "all fields of an edited asset can be modified except Ids", which do not correspond to use cases. The approach could be extended by adding recognition of these different forms of requirement, e.g., by classifying requirements statements.

On the example sentence, the approach produces the result:

```
class doctor { }
usecase viewpatient'ehr
{ actor = doctor; }
```

The use case is correct, but the name of the use case contains an invalid character.

2.2 Elallaoui, Nafil and Touahni Approach (Elallaoui et al., 2018)

In contrast to the preceding approach, this approach takes as input semi-structured user stories, and applies POS-tagging as its main NLP technique. The input text sentences are POS-tagged and the words of each sentence are filtered to remove adjectives and auxiliary words, retaining only nouns and verbs. The first noun/compound noun in each sentence is assumed to be the actor of the use case. The following verbs and nouns then make up the action of the use case.

The evaluation uses a single case, but of large size (168 user stories). Recall and precision compared to a manually-constructed use case model are calculated, with high precision and recall values for actors ($p = 98\%$, $r = 98\%$), use cases ($p = 87\%$, $r = 85\%$) and their relationships ($p = 87\%$, $r = 85\%$).

On the example sentence, the approach produces the result:

```
class doctor { }

usecase wishviewpatientEHR
{ actor = doctor; }
```

This is a valid formalisation.

2.3 AgileUML (Lano et al., 2021)

This approach operates on either unstructured or semi-structured behaviour specifications. It performs segmentation into sentences and POS-tagging, and uses a decision tree classifier to distinguish sentences that express user stories from those that express data requirements or general constraints. Both class definitions and use cases are derived from the classified sentences, using heuristics to recognise the class names, attribute names, actors and actions in the text. A thesaurus/glossary is used to classify words/phrases. Approximate matching using text edit distance (Levenshtein et al., 1966) is used in order to allow for variation in word form.

The evaluation is performed on 27 cases, including 24 real-world cases. There are 10 large cases (over 75 user stories), 2 small and 15 of medium size (25 to 74 use cases). The average F-measure is 94%, based on comparison of the automatically formalised models with manually-derived models.

On the example sentence, the approach produces the result:

```
class Doctor {
    stereotype originator="1";
}

class Patient {
    stereotype originator="1";
}

usecase viewThePatient : void {
    parameter doctorx : Doctor;
    parameter patientx : Patient;
    stereotype originator="1";
    stereotype actor="Doctor";
    stereotype read;

    ::
        true => patientx->display();
}
```

Here, tracing information is embedded into the model using the *originator* tag. Executable behaviour is produced for use cases where possible, so that they

can be immediately used for prototyping. However the key noun ‘EHR’ is missing from the use case name.

The authors find that a major cause of poor formalisation results is incorrect tagging and incorrect parsing by the NLP tools used (Stanford NLP and Apache OpenNLP). Thus the formalisation algorithms fail because the input they are given is semantically incorrect (e.g., ‘existing’ mis-classified as a verb in a phrase ‘the existing files’).

2.4 Simple Heuristic Approach

This approach operates on semi-structured user stories as input. It tokenises the input sentences and applies POS-tagging. For each sentence it attempts to recognise the entities (classes) referenced in the sentence as those noun phrases $DT?JJ*NV+$ which have a noun in a predefined glossary of ‘entity’ nouns. Use cases are recognised from those sentences which contain both a verb and a modal verb. Chunking of the sentence according to the pattern $[\wedge VB]*VB*MD[\wedge VB]*VB[\wedge VB]*$ is performed, where the first block of non-verbs is used to form the actor name, and the second block starting with the first verb following the modal verb is taken as the use case action. Finally, the category of this verb is used to classify the kind of use case as ‘create’, ‘edit’, ‘read’, ‘delete’ or ‘other’.

For example, “As a doctor, I wish to view the patient’s EHR” would be chunked as [As, a, doctor, I], [wish], [to], [view, the, patient’s, EHR]. This would produce the use case

```
usecase view_the_patient_EHR {
    stereotype actor="doctor";
    stereotype "read";
}
```

Doctor and *Patient* would become classes.

2.5 Summary

The approaches all use POS-tagging and segmentation of the text into sentences as initial steps. They differ in their strategies for extracting use case elements from the resulting text, although some form of chunking based on the expected form of behavioural requirements is an essential part of each strategy. The evaluations of each approach all use the concepts of accuracy based on recall/precision and F-measure, but the number, scale and provenance of evaluation examples differ, as does the basis for computing accuracy. This approach to compute accuracy also depends upon the subjective judgement of the evaluator as to whether a formalised element is correct or not.

3 EVALUATION FRAMEWORK

In order to provide a platform for consistently comparing different requirements formalisation approaches, we define an evaluation framework which consists of:

- A domain-specific language (DSL) for expressing NLP pipelines
- An instantiation of the DSL in Python, using the Python NLTK NLP library, together with a specific library of utility functions for requirements formalisation
- A set of evaluation examples taken from real-world requirements documents, together with manually-derived ‘gold standard’ formalised models produced by a rigorous process
- Evaluation tools, to perform a threefold evaluation of the models created by each RF approach (Figure 1).

This framework has the benefit of a high degree of automation: the evaluations can be performed without human subjectivity entering into the assessment, as could occur if precision/recall figures are estimated based on manual comparison of two models.

The DSL includes statement constructs for loading datasets, filtering and transforming datasets, and performing analysis operations on them, and for saving datasets. The syntax is based on SQL. A novel facility is the ability to specify chunking transformations by regular expressions. Thus a regular expression formed from POS names or generalised POS names can be written in order to specify that a POS-tagged text is to be split into chunks that match the expression.

The evaluation examples for user stories include 1 large (FABSucs) and 1 medium sized example (k3ucs) of user story specifications, taken from different sources (Kaggle, 2021) and (Mendeley, 2021), and written using different styles. There are also evaluation examples for unstructured data requirements, including a real-world requirements statement case.

The evaluation tools are as follows:

- *checkModelNames.py* – checks the names of model elements, i.e., whether attributes, use cases and classes have valid names, including a check for duplicate names and a check that class names should be a singular noun. This check helps to ensure that the models are suitable for use as the specification for an application.
- *compareModel2Source.py* – checks the percentages of source document nouns and verbs which also appear in the generated models. This helps to ensure that all information from the source document has been represented in the formalised model.

- *compareModels.bat* – compares the reference ‘gold standard’ model for a case to the model produced by a formalisation approach. This generalises the usual precision/recall estimates by (i) comparing classes and attributes in the two models, in addition to use cases; (ii) allowing partial matches between names of elements, based on string edit distance.

The tools may be accessed at (Lano, 2023).

We also include the F-measure accuracy estimation, based on the standard definition but with a 0.5 correctness score for an element which is identified but with significant differences to the correct version (e.g., its name includes several superfluous words or omits necessary words).

We do not evaluate the efficiency/time performance of approaches because these are implemented on different platforms and hence comparing execution times would not give information about the intrinsic efficiency of the approaches.

4 EVALUATION OF APPROACHES

We apply the three comparisons of Figure 1 to each of the approaches of Section 2, for the k3ucs and FABSucs requirements cases. We also estimate our modified F-measure accuracy for recognising use cases. All artefacts and results of these evaluations may be accessed at (Lano, 2023).

Tables 1 and 2 show the model quality scores for each approach and the two evaluation cases.

Table 1: Formalised model quality: k3ucs.

Approach	Class validity	Attribute validity	Use case validity	Flaws
<i>Hamza/Hammad</i>	0	0	1	0
<i>Elalloui et al</i>	0.33	0	0.42	17
<i>AgileUML</i>	0.88	1	1	3
<i>Simple heuristic</i>	0.95	0	1	1

Table 2: Formalised model quality: FABSucs.

Approach	Class validity	Attribute validity	Use case validity	Flaws
<i>Hamza/Hammad</i>	0	0	1	0
<i>Elalloui et al</i>	0.57	0	1	6
<i>AgileUML</i>	0.96	1	1	3
<i>Simple heuristic</i>	0.96	0	1	2

Tables 3 and 4 show the model completeness scores for each approach and the two evaluation cases.

Tables 5 and 6 show the model accuracy scores for each approach and the two evaluation cases.

Table 7 shows the modified F-measure for use case recognition for each approach and both cases.

Table 3: Formalised model completeness: k3ucs.

Approach	Noun completeness	Verb completeness
<i>Hamza/Hammad</i>	0	0.4
<i>Elalloui et al</i>	0.08	1.0
<i>AgileUML</i>	0.51	1.0
<i>Simple heuristic</i>	0.45	0.93

Table 4: Formalised model completeness: FABSucs.

Approach	Noun completeness	Verb completeness
<i>Hamza/Hammad</i>	0	0.32
<i>Elalloui et al</i>	0.09	1.0
<i>AgileUML</i>	0.41	0.86
<i>Simple heuristic</i>	0.36	1.0

Table 5: Formalised model accuracy: k3ucs.

Approach	Class similarity	Attribute similarity	Usecase similarity
<i>Hamza/Hammad</i>	0	0	0.47
<i>Elalloui et al</i>	0	0	0
<i>AgileUML</i>	0.04	0.002	0.68
<i>Simple heuristic</i>	0.007	0	0.57

4.1 Discussion

The accuracy of all the approaches, as estimated by model similarity (Tables 5, 6) is below 70%, which indicates that they need to be used in conjunction with human expertise to refine or correct their results, and that they do not provide a fully automated solution. The formalisation completeness results for nouns is quite low (Tables 3 and 4), indicating that information on data elements is being ignored or lost by the formalisation processes. The completeness for verbs is generally high except for the Hamza/Hammad approach. On inspection the low results for this approach are due to verb stemming, so that the original version of the verb is lost and only the stem retained in the resulting model. On the other hand, the AgileUML approach adds the purpose of the user story into the use case name, resulting in excessively long and complex names. The simple heuristic approach sometimes adds the actor part of the user story into the use case name: in cases such as “The system should allow a staff member to ...”, the actor is wrongly assigned as ‘The system’. The Ellaloui et al approach also has the same flaw. It should be a user-configurable choice whether certain terms such as ‘System’ or ‘Application’ can be accepted as actors.

Although the Hamza/Hammad and AgileUML approaches aim to recognise a range of different textual formats for user stories, there are still some cases which they fail to process correctly. None of the approaches are able to create use cases involving two or

Table 6: Formalised model accuracy: FABSucs.

Approach	Class similarity	Attribute similarity	Usecase similarity
<i>Hamza/Hammad</i>	0	0	0.6
<i>Elalloui et al</i>	0.01	0	0.56
<i>AgileUML</i>	0.02	0	0.75
<i>Simple heuristic</i>	0	0	0.57

Table 7: F-measure for cases.

Approach	k3ucs	FABSucs
<i>Hamza/Hammad</i>	0.23	0.15
<i>Elalloui et al</i>	0.35	0.23
<i>AgileUML</i>	0.69	0.81
<i>Simple heuristic</i>	0.71	0.55

more actors, instead the AgileUML approach creates use cases which can be linked to several entities via data usage relationships.

All of the approaches use heuristic rules to recognise use case elements. The only explicit use of machine learning (ML) is the decision tree classifier used by AgileUML to distinguish different categories of requirements. It may be difficult to use ML to learn the derivation from user stories to use cases because of the relatively small amounts of data available for training.

The use of tools such as WordNet or Word2Vec (Mikolov et al., 2013) to compute word similarity scores could also be of potential use to improve the approaches.

5 RELATED WORK

Since 2018 there has been a noticeable increase in the number of papers that apply NLP and DL techniques for automatic generation of UML diagrams from requirements.

The automation of natural language analysis for extraction of UML diagrams is emphasized in (M. Maatuk and A. Abdelnabi, 2021). The main focus of this work is to apply NLP techniques and heuristic rules to generate usecase and activity diagrams. The Stanford CoreNLP tool is used to perform NLP tasks. Following these, heuristic rules are applied individually for generation of activity and usecase diagrams. In (Xu et al., 2020) a utilities permitting system based on an NLP algorithm is designed to formalise the requirements of road agencies in UML and OCL formats. The input requirements are in textual format. The NLP process includes a pre-processing step, which tokenises and splits sentences. In this step words are classified into parts of speech and labeled according to POS tags. Then occurrences of the terms in the sentences are recognised and in the third step by applying a

chunking technique the sentence structure is analysed and represented as tree structures. Finally, five rules are applied to generate target information from tree structures. The system is validated in terms of performance and applicability by using random cases. The Requirement Transformation (RETRANS) approach is presented in (Kamarudin et al., 2015), this generates usecase and activity diagrams from requirements in text format by applying model transformation and NLP techniques. An NLP algorithm is designed in (Alashqar, 2021) to generate sequence and class diagrams from scenario-based requirements. A software tool called automatic generation of UML (AGUML) is presented to perform all the tasks automatically. Experimental results are reported to show the applicability and performance of the approach. In (Sanyal et al., 2018) an automatic approach for generation of class diagrams from semi-structured text inputs is presented. Some keywords are used to structure the input. NLP techniques and heuristic rules are used to generate the result by applying the procedure in four steps. In the first step classes are extracted followed by generation of attributes in the next step. Following that methods and relations are extracted. The last three steps depend on the first step in this procedure.

A ML approach for extraction of classes and attributes from unstructured plain text is introduced in (Elmasry et al., 2021). Two classifiers are used to classify each word into class and attributes. To relate appropriate attributes to classes dependency parsing is used. A public requirements dataset is used throughout this research (Ferrari et al., 2017). NLP techniques are used in the pre-process and post-tagged phases to transfer data to the ML tasks. Then the machine learning algorithms of Support Vector Machine (SVM) and Naive Bayes (NB) are applied for extracting classes and attributes. A text-to-model transformation framework for mapping textual requirements to UML models is presented in (Sedrakyan et al., 2022). The authors emphasize on integrating machine learning methods, word embedding, heuristic rules, statistical and linguistic knowledge to increase the quality of the outcome. A web application for generation of use case and class diagrams from English text is presented in (Narawita et al., 2017). In this research NLP and ML techniques are used. Tokenization, POS tagging, chunking and splitting are NLP techniques that are applied in this process. Finally, the visual representation of diagrams are provided using Visual studio. RF using LLMs is investigated with ChatGPT by (Camara et al., 2023). The results show that some basic requirements formalisation ability is present in ChatGPT, however specific fine-tuning of a suitable LLM (pre-trained with datasets including software models)

by instruction training for the formalisation task would be necessary to improve this capability.

Although different works have been investigating the concepts of RF by applying NLP and DL techniques, however this field remains at an experimental stage. Most of the works in this domain applied heuristic approaches, however these were not evaluated on a broad range of input cases. Therefore, it is not possible to determine the applicability of such approaches in specific domains. Only a few approaches have applied DL in the RF domain, this limited use is likely to be due to the limited quantity of available appropriate training data (i.e., relating requirements text to formalised models). In general most of the ML approaches do not apply the whole potential of DL in the domain. Furthermore, there is no standard benchmark and evaluation criteria for comparing different RF cases. Therefore, in this research we provide a set of requirement statement to compare the effectiveness of RF approaches according to the standard criteria.

6 FUTURE WORK

We intend to enlarge the set of evaluation cases to include a wider range of requirements documents, and to expand the evaluation to more approaches, including data-oriented formalisation (creating class diagrams). Formalisation approaches for unstructured and structured requirements documents will also be evaluated.

Further evaluation criteria could be added, for example, some measure of how configurable and adaptable an approach is: to what extent it permits users to modify any parameters, strategies or knowledge bases used in its formalisation process. Another form of comparison could be a ‘blindfolded taste test’ where a group of independent software engineers evaluate and rank alternative formalisations of a requirements statement, without knowing the identity of the approach which produced the formalisation.

7 CONCLUSIONS

We have provided the first framework for systematic and objective comparison of requirements formalisation approaches, and demonstrated the application of this framework to compare four alternative approaches to behavioural requirements formalisation. The results provided useful insights into the issues and factors which such approaches need to address to produce useful formalisations.

REFERENCES

- Alashqar, A. M. (2021). Automatic generation of uml diagrams from scenario-based user requirements. *Jordanian Journal of Computers and Information Technology*, 7(02, June 2021).
- Apache (2021). Apache opennlp toolkit. <https://opennlp.apache.org>. 2021.
- Camara, J., Troya, J., Burgueno, L., and Vallecillo, A. (2023). On the assessment of generative ai in modeling tasks. *SoSyM*, 22.
- Chen, M. et al. (2021). Evaluating large language models trained on code. *arXiv preprint*, 2107:03374v2.
- Elallaoui, M., Nafil, K., and Touahni, R. (2018). Automatic transformation of user stories into uml use case diagrams using nlp techniques. *Procedia computer science*, 130:42–49.
- Elmasry, I., Wassif, K., and Bayomi, H. (2021). Extracting software design from text: A machine learning approach. In *2021 Tenth International Conference on Intelligent Computing and Information Systems (ICI-CIS)*, pages 486–492. IEEE.
- Ferrari, A., Spagnolo, G. O., and Gnesi, S. (2017). Pure: A dataset of public requirements documents. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 502–505. IEEE.
- GitHub.com (2022). GitHub CoPilot, <https://copilot.github.com/>.
- Guo, D. et al. (2021). GraphCodeBERT: Pre-training code representations with dataflow. In *ICLR 2021*.
- Hamza, Z. A. and Hammad, M. (2019). Generating uml use case models from software requirements using natural language processing. In *2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO)*, pages 1–6. IEEE.
- Kaggle (2021). Kaggle software requirements dataset. <https://www.kaggle.com/iamsouvik/software-requirements-dataset>. Accessed: 2021.
- Kamarudin, N. J., Sani, N. F. M., and Atan, R. (2015). Automated transformation approach from user requirement to behavior design. *Journal of Theoretical and Applied Information Technology*, 81(1):73.
- Kolahdouz-Rahimi, S., Lano, K., and Chenghua, L. (2023). Requirement formalisation using natural language processing and machine learning: A systematic review. *International Conference on Model-Based Software and Systems Engineering*.
- Lano, K. (2023). Requirements formalisation repository. <https://www.https://github.com/kevinlano/RequirementsFormalisation>. Accessed: 2023.
- Lano, K., Yassipour-Tehrani, S., and Umar, M. (2021). Automated requirements formalisation for agile mde. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 173–180. IEEE.
- Levenshtein, V. I. et al. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710.
- M. Maatuk, A. and A. Abdelnabi, E. (2021). Generating uml use case and activity diagrams using nlp techniques and heuristics rules. In *International Conference on Data Science, E-learning and Information Systems 2021*, pages 271–277.
- Mendeley (2021). Mendeley user story dataset. <https://www.data.mendeley.com/datasets/bw9md35c29/1>. Accessed: 2021.
- Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
- Narawita, C. R. et al. (2017). Uml generator-use case and class diagram generation from text requirements. *The International Journal on Advances in ICT for Emerging Regions*, 10(1).
- Otter, D. W., Medina, J. R., and Kalita, J. K. (2023). Requirement formalisation using natural language processing and machine learning: A systematic review. *Model-sward 2023*.
- Princeton University (2021). Wordnet. <https://www.wordnet.princeton.edu>. Accessed: 2021.
- Santorini, B. (1990). Part-of-speech tagging guidelines for the penn treebank project. University of Pennsylvania, School of Engineering and Applied Science.
- Sanyal, R., Ghoshal, B., et al. (2018). Automatic extraction of structural model from semi structured software requirement specification. In *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, pages 543–58. IEEE.
- Sedrakyanyan, G., Abdi, A., Van Den Berg, S. M., Veldkamp, B., and Van Hillegersberg, J. (2022). Text-to-model (tetomo) transformation framework to support requirements analysis and modeling. In *10th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2022*, pages 129–136. SCITEPRESS.
- Stanford University (2020). Stanford nlp. <https://www.https://nlp.stanford.edu/software/>. Accessed: 2020.
- Xu, X., Chen, K., and Cai, H. (2020). Automating utility permitting within highway right-of-way via a generic uml/ocl model and natural language processing. *Journal of Construction Engineering and Management*, 146(12):04020135.
- Zhao, W. et al. (2023). A survey of large language models. *arXiv*, 2303.18223v10.