JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 11, No. 1, March 2026, Pp.637-643

DRE!: A RULE-BASED SOFTWARE REQUIREMENT EXTRACTION FRAMEWORK FOR INDONESIAN DOCUMENTS

Moh. Zulfiqar Naufal Maulana*[1)], Ahmad Reza Adrian[2)], Fathan Alfariel Adhyaksa[3)], Didik Dwi Prasetya[4)], Jevri Tri Ardiansyah[5)], Hanif Rifai Adha[6)]

1. Informatics Engineering, Faculty of Engineering, Universitas Negeri Malang, Indonesia
2. Informatics Engineering, Faculty of Engineering, Universitas Negeri Malang, Indonesia
3. Informatics Engineering, Faculty of Engineering, Universitas Negeri Malang, Indonesia
4. Informatics Engineering, Faculty of Engineering, Universitas Negeri Malang, Indonesia
5. Informatics Engineering, Faculty of Engineering, Universitas Negeri Malang, Indonesia
6. Electrical Engineering, Faculty of Engineering, Universitas Negeri Malang, Indonesia

Article Info
Keywords: dependency parsing; functional requirements; natural language processing; requirements extraction; software engineering

Article history:
Received 25 October 2025
Revised 4 March 2026
Accepted 12 March 2026
Available online 12 March 2026

DOI :
https://doi.org/10.29100/jipi.v11i1.9357

* Corresponding author.
Moh. Zulfiqar Naufal Maulana
E-mail address:
zulfiqar.naufal.ft@um.ac.id

ABSTRACT

The process of deriving software requirements from descriptive documents is a crucial one for software engineering. Unfortunately, this can be a laborious and time-consuming process when done by hand. We present DRE! In this work, a framework for the automatic extraction of functional requirements from system descriptions in an Indonesian based language. A structured workflow for the proposed approach The system is then responsible for data acquisition based on a predefine list of actors. (then take some preprocessing and parse the sentence structure using word dependency) One important component in DRE! is the actor detection phase. During this stage, the system can also look back to earlier sentences for context. Afterwards actor based templates are applied to generate the requirements extracted. Similarity between requirement statements is calculated using cosine similarity to eliminate redundancy. We evaluated DRE! using eight diverse datasets from the software description field. The F1 scores for all but one of the eight datasets are in the range between 0.76 and 0.88, which indicates good performance. The best case result was achieved on the "EduLearn" dataset with a score of F1 "0.88". However, it performed lower on the "LogiWare" dataset (F1 score of 0.52), which can mainly be assigned to a high amount of false positives. These findings suggest that although DRE! can well serve the purpose of automating requirements extraction its performance is ultimately dependent on the general clarity and style of writing in the input document.

I. INTRODUCTION

In this digital era, software is basically the backbone for almost everything we do, from banking and healthcare to the apps on our phones. It's the main thing that drives innovation and makes business operations more efficient [1]. However, we are evolving these systems with increasing complexity by the day; a random approach will only get us so far. This is the reason why we require a systematic and disciplined way of working in order to ensure that the development is successful. Such fundamentals are why Software Engineering (SE) is absolutely necessary and is what gives skeleton to complex requirements, construction of a system to reach the technical target [2], [3].

The requirements engineering (RE) phase is more than the figure, as it belongs to the software development life cycle. Failure to identify requirements accurately in this stage, can create 'snowball effect' kind of scenarios leading to over-budgeting, delays or complete flop of the project [4]. In a gist, quality of a software product often comes down to how well the team really understands what they are actually building [5]. The task of discovering and recording this information is, we refer to true pieces as requirements elicitation [6], quite complex due to the fact that it flows from different players in a rather obscure form.

In order to address these problems, an increasing number of SE researchers are trying to automate requirement extraction from documents such as system descriptions [6]. However, there are many tools and research in this area that is only available for English because they have the largest amounts of datasets and linguistic resources. It's a different story in Indonesia. Actually, there are very few public datasets available for software requirements

extraction [7]. Filling this gap is a significant challenge for local researchers and practitioners that want to create or test extraction methods that actually work in our context.

There are. In fact, several studies have already been conducted in the Indonesian context. For example, Haris et al. [8] provide an approach that works for automated features extraction from SRS (Software Requirement Specification which is the very nerve of SPLE (Software Product Line Engineering). One more interesting study was conducted by Raharjana et al. [9] examined deriving requirements in the User Story format (the who, what, and why do you remember?) but they used online news as their source instead of normal documents. In both of these studies, rule-based methods achieved the desired results using POS Tagging and Dependency Parsing.

While some of the techniques are shared, our study is actually quite separate since we apply these linguistic tools. First, the DRE! framework relies on formal system description documents as the source of data, which is orders of magnitude more structured than anything like online news. Second, we aim to create actor-oriented functional requirements based on boilerplate process rather than feature extraction targeting for SPLE or user story creation. Third, we accomplish actor detection with an existing list augmented by context-infused in the form of previous sentences. Contrast this with the NER-based methods employed in those other studies.

That's why we research how to build a DRE! namely a framework that we designed, specifically for extracting functional requirements from system descriptions in Indonesian automatically. We try to offer two main things by using a linguistics analysis rule based approach. I) First, we would like to offer the means to automate the requirements elicitation process. Secondly, we are hoping this could be a starting point to tackle the shortage of resources for software engineering text processing for Indonesian language.

II. RESEARCH METHODS

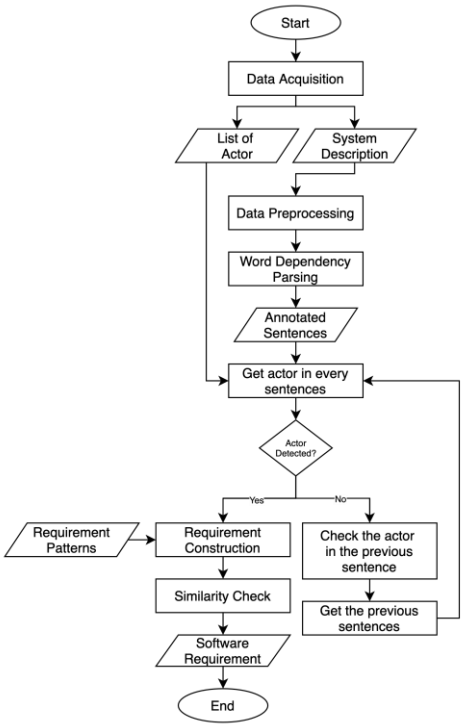

Fig 1 DRE! Framework

We follow a multi-step, systematic workflow to extract functional requirements as part of our research methodology. (1) Data Acquisition The process begins by preparing a system description document and listing the actors involved Then, (2) Data Preprocessing is performed for cleaning the text and then (3) word dependency parsing. The most critical part of the System is (4) Actor Detection. In this case, the system reads through every sentence looking for an actor from our list. If it doesn't find one from the context, we added a mechanism to check the previous sentence for a contextual reference. The system, once identified an actor, proceeds to the next step (5)Requirement Construction where it maps a sentence against certain Requirement Patterns that can further breakdown

into structured content. Lastly, we perform step (6) Similarity Checking. So, make sure this exercise is held to avoid repetitive requirements and have a clean output of Functional Requirements in the end.

A. Data Acquisition

We used the public dataset called 'Indonesian Software Descriptions' as our data source for this study [10]. This particular dataset was chosen because it aligns best with the Indonesian-language context. Also, their annotations are pretty substantial, so it was really beneficial for our research.

TABLE 1
DATA OVERVIEW OF INDONESIAN SOFTWARE DESCRIPTION DATASET

Label	#Actor	#Requirements
BuildTech	3	25
LogiWare	2	11
Quickship	5	15
EduLearn	3	17
EduTech	2	13
EduSkill	3	15
BeautyCare	3	13
FastMed	3	13
Total	**24**	**122**

TABLE 1 provides information on the 'Indonesian Software Descriptions' dataset [10]. We selected eight text documents from the collection to use in our study. We selected these particular documents to achieve a good balance between document length. We also wanted to ensure that there is enough variation in the total number of requirement sentences, which we use as our groundtruth.

These descriptions are used as raw input which our system uses to extract requirements. The original researchers labeled software requirements, a total of 122 sentences (threats), as these were not present in the initial prediction. We treat this collection as our ‘gold standard’ or ground truth. Essentially, it’s what we utilize for measuring and evaluating how accurate our framework’s extraction results actually are.

B. Data Preprocessing

The raw software descriptions that will be extracted still go through a series of steps to prepare, clean and structure the text before we start extracting data. This preprocessing stage is really significant since it directly influences the quality of our analysis as well as the accuracy of the model. To begin with, we perform case folding which means transforming all letters in the documents to lower-case. The here goal is simple, we want to standardized the data such that same word with different capitalization i.e, 'konstruksi' and 'Konstruksi' have treated as exact same token. Then we pass the words through POS Tagging which analyzes each word in a sentence and assigns it a label based on its corresponding part of speech like Noun(NN), Verb(VB), Adjective(JJ) [11]. This process tags the data with structural information. This POS tag info is so critical for the next things, finding out the syntactic patterns (such as specific verb pattern) pointing to a need.

C. Word Dependency Parsing

Word Dependency parsing follows after preprocessing. This means that the aim here is to parse or to analyse each sentence grammatically into syntactics, so that the system understands how those words can relate to one one another‹ [12]. Basically, this meth Essentially, this technique outlines the asymmetric correlation between a head word (the governor) and its dependentsod maps out the 'asymmetric' relationship between a 'head' word (the governor) and the words that depend on it, called 'dependents' [13]. This results in what’s known as a dependency tree, which expresses the grammatical function of each word, like how a subject is connected to its main verb (the root) and one an object.

For example, if you have the phrase Admin dapat mencetak laporan bulanan (Admin can print the monthly report), this parser will find that Admin is a nominal subject (nsubj) on verb mencetak (print) and laporan (report) as its direct object (dobj). What I mean with the structure detection is that the system can actually get what each word in a sentence means and what context it belongs to, rather than just read it as a list of words. The output generated here is a set of tagged sentences that we use as the main input for the next step.

D. Get Actor in Every Sentences

Once we have all these annotated sentences from the parsing stage, can go into Actor Detection. The next step simply has the system examine sentence by sentence to find an actor. We do this through the preconfigured actor list we created during data preparation.

JIPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)
Journal homepage: https://jurnal.stkippgritulungagung.ac.id/index.php/jipi
ISSN: 2540-8984
Vol. 11, No. 1, March 2026, Pp.637-643

So what do you do if a sentence doesn't specifically name an actor? To handle this we built a simple referencing system. The system scans the last sentence o learn which subject is being mentioned and infers that this same subject is now the actor of this new sentence as well. This is an essential step, because it makes sure that every possible sentence has a specific subject in mind when we move on to the next part of the process.

E. Requirement Construction

Once any sentence is connected to an actor, it enters the final stage: Requirement Construction. Here the goal is to refine these candidate sentences into a well-structured complete coherent set of functional requirements.

In this case, we used a modified version of Rupp's boilerplate to frame our software requirements [14] Fig 2. According to weakest one or/or higher level attacker of 'The System' by default, Subject of requirement will always be system Evoker in template of Rupp. In our case though, we wanted something more flexible.

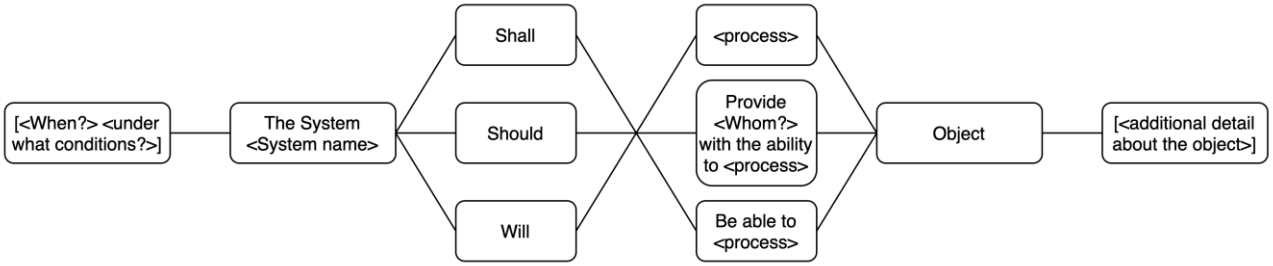

Fig 2 Rupp's Boilerplate

But to retain the usefulness and clarity of that template, we went for an actor-oriented adaptation (shown below in Fig 3). The most significant change we made was replacing the fixed 'The System' subject with dynamic placeholders for the Actor, [9], [15], [16], [17]. This modification goes hand in hand with the logic of our workflow; the system just needs to take this actor info it found during the previous detection on each image and fill out this placeholder. Then the actors are put in their right spot in the final requirement sentence format.

One of the really nice things about this actor-centric approach is that it has several big advantages. To begin with, it makes its requirements statements more direct and to the point [18], [19], [20]. Second, it's much easier to read and understand the context as the reader can see immediately who is actually using or benefiting from the functionality [21], [22]. Lastly, this format is spot on with cutting edge software development trends e.g. User Stories in Agile. In the same way as those techniques, we put the user, or actor, at the heart of our requirement.

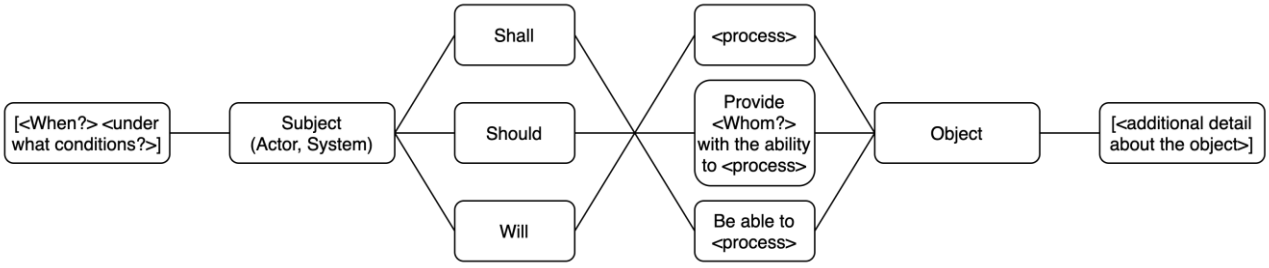

Fig 3 The Rules of DRE! Framework

F. Similarity Check

In NLP, we use Cosine Similarity as a metric to determine the similarity of two sentences. This is interesting, because this does not even take into account the length of the sentences; rather, it calculates the cosine (angle) between two vectors in multidimensional space. From this we get some score from 0 to 1. A score of 1 indicates that the two sentences are essentially identical and a score of 0 means they don't share any words.

For real-world applications, setting an appropriate threshold on the cosine similarity is somewhat context-dependent. In some situations, 0.1 to 0.15 is all that is necessary. In fact, in an identification system for comparing tattoo images the researchers applied a threshold value (0.15) to avoid that potential matches would be missed; In this particular case they reviewed images that surpassed this score, which led to recall rate 1.0 and a reduction of manual labour of 20% [23]. Similarly, a book recommendation system used dates > 0.1 to filter relevant books and attained precision 0.7 and recall of 0.73 [24]. On the other hand, in case of security or authentication, this threshold

becomes a highly sensitive parameter. Testing and experimental to find out the false positive, versus the false negative ratio can often result in achieving a better outcome [25].

In order to do so, in this study we set up a similarity threshold of 0.45 to remove identical requirement sentences for redundancy . This means that the system will treat two sentences as duplicates if their similarity score is above this threshold. When it does so, it keeps one by default and discards the rest. As a result, the list of requirements can contain much less duplicitous information. We make sure to perform this filtering right before transforming an individual unique sentence into the requirement template during the final construction step.

III. RESULTS AND DISCUSSION

This chapter presents the main results of our research on automating the extraction of software requirements. In Section 6.3, we intentionally applied the proposed system on some datasets of software description to extract functional requirements as discussed in previous chapter. We will start with some of the quantitative findings. We evaluated the model with four common metrics to see how well it didn't really do: Accuracy, Precision, Recall and F1-Score. These metrics allow us to determine how correct and complete our extractions according to the ground truth.

We'll drill into a more detailed conversation after we cover some numbers. But first we will decode what those metric scores actually mean about the quality of our method. Next, we will analyze several qualitative case studies. We'll look at how well the system got things right (True Positives) and where it failed (False Positives, False Negatives). So this error analysis is not even a bad thing because it allows us to take a deeper insight in the weaknesses and strengths of our current workflow.

A. Results

This graph summarizes the diversity of text volumes our system had to deal with and displays the differences between our eight software datasets for description lengths in *Fig 4*. Vertical axis (Y) represents the absolute number of words per document and horizontal axis (X) corresponds to name of each software.

From the chart, it is clear that the lengths of descriptions differ from dataset to dataset. For example, 'BeautyCare' with the longest description of contents in 449 words and 'LogiWare' with 428 words. At the other end of the scale, 'BuildTech' is also the shortest 355 words.

The rest of the datasets is somewhere in between. 'EduLearn' is 416 words, 'Quickship' has 393. After this, there are also 'FastMed', 'EduTech' and 'EduSkill', to name a few, which come in quite similar length between 382 words and down to 365 words. On the whole, it turns out that our test data is not one-size-fits-all, with a settling of almost 100 words between the longest and shortest theory.

Between the opening act and DRE's performance! In the proposed framework, quantitative findings are presented in TABLE 2. This table summarises the model performance over all eight datasets. The first lists the test cases and each row is dedicated to a specific case. The first column (#Req) gives the true number of functional requirements in that document (ground truth), while the second one (#Extracted) the number of requirement candidates actually returned by the system.

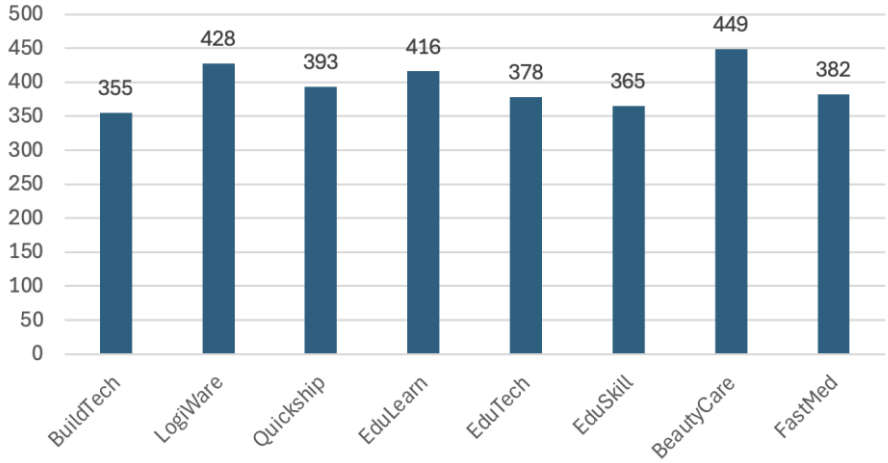

Fig 4 Diversity of Software Description Length

Next, we evaluated the model performance using Confusion Matrix-based standard metrics. This matrix records three things; True Positives(TP)–the actual requirements we mined, False Positives(FP) sentences the system mistakenly identified as a requirement and False Negatives (FN) -genuine requirements which did not get captured by the system. Four metrics were calculated (Accuracy, Precision, Recall and F1-Score) based on the above values.

For instance, the following is an example with the 'BuildTech' dataset. The 25 real requirements, were extracted in 23 sentences by our systems. If we dig a little deeper into the results, there are 20 TPs, 5 FPs and 3 FNs. Calculating the numbers, this yields us a Precision equals to 0.87 Recall equals to 0.80 and an F1-Score equal to 0.83 So we did this for the other seven datasets in order to summarize how the system behaved in each specific scenario.

TABLE 2
RESULT EXTRACTION OF DRE! FRAMEWORK

Label	#Req	#Extracted	Confusion Matrix			Accuracy	Precision	Recall	F1-Score
			TP	FP	FN				
BuildTech	25	23	20	5	3	0.71	0.87	0.80	0.83
LogiWare	11	20	8	12	3	0.35	0.40	0.73	0.52
Quickship	15	14	11	3	4	0.61	0.79	0.73	0.76
EduLearn	17	19	15	2	2	0.79	0.88	0.88	0.88
EduTech	13	15	11	2	2	0.73	0.85	0.85	0.85
EduSkill	15	17	13	2	2	0.76	0.87	0.87	0.87
BeautyCare	13	16	12	3	1	0.75	0.80	0.92	0.86
FastMed	13	18	13	5	0	0.72	0.72	1.00	0.84

B. Discussion

TABLE 2 It shows results, and confirms that the method achieves a reasonable performance although clearly this works much better with well written descriptions. Both models were learned on F1-Scores whose performance received between 0.76 to 0.88 in the case of 7 from 8 databases. `EduLearn' was the most successful one reaching a fine 0.88 for Precision, Recall and F1-Score. Having said that, when the planets align, this model is both accurate and complete at the same time.

But we discovered a prominent outlier as well: The 'LogiWare' dataset scored an F1−Score of just 0.52. Digging deep into the confusion matrix, we problem down to.low Precision (0.40). From the 20 sentences that were extracted by the system, only 8 were indeed valid requirements (TPs), while 12 counted as false positives (FPs). In this case, the model seems to have been too ambitious, marking non-operational sentences as requirements. This likely happened, because the 'LogiWare' text is slightly more on the 'nervous' side that it can probably use words such as 'can' or 'will' in contexts that aren't necessarily requirements and might have tricked our pattern matching rules.

Now let’s see another interesting case, ‘FastMed’ has 1.00 Recall score. This implies that no requirements were missed by the system (FN=0). But it did come with a cost: It also had 5 false positives, so the precision was down to 0.72This really highlights the traditional tug of war between Precision and Recall. Our rules are relatively broad, which is great to not leave anything on the cut floor but this does come with a risk of putting in junk sentences too.

Ultimately, these results illustrate that source document clarity is paramount in this case. Performance, oddly as we noted in Fig 4, does not seem to correlate with length of document. 'BeautyCare' document was the longest across all of these, yet's it values for F1-Score remains in good place: 0.86. The implication is that it’s the nature of the narrative and lack of ambiguity that really drive how well the model performs.

IV. CONCLUSION

This study was centered on doing construction and testing of DRE!, which is used to facilitate the extraction of a functional aspect of the requirements from Indonesian descriptions of systems. The results of our evaluation on eight different datasets show that the methodology works well and achieves its main goals. This enables the framework to efficiently identify and convert candidate sentences into a structured, actor-based representation.

Our primary result indicates that a rule-based approach in conjunction with linguistic analysis, such as dependency parsing, and a list of pre-defined actors should be an effective strategy (shown in Tables 3 and 4). The high F1-Scores we noticed in a lot of our searches confirm this. LogiWare taught us that the main weakness in the system lay behind how document is written. We learned that the quality of the narrative, not documentation length, is what truly matters with respect to accuracy. We hope that, in the end, this research provides a useful workflow to reduce the manual work that istypically required for requirements elicitation.

There's plenty of scope to improve the system, however; while the results are a good start, they could be even better. In terms of future work, we foresee two major improvements: (1) To decrease False Positives found in 'noisy' texts, further studies could use machine learning or deep learning models (e.g., BERT). This would allow the system not only to perform simple pattern-matching, but also to focus on distinguishing general descriptions from actual requirements. (2) A hybrid approach would be the next step as our current actor detection is dependent only on a predefined list. This information allowed us to essentially filter out cases that - while they may be of interest from a NER perspective - simply didn't have key actors and clearly weren't going to happen when we ran them against the list.

REFERENCES

[1] I. Ahmed *et al.*, "Artificial Intelligence for Software Engineering: The Journey So Far and the Road Ahead," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, pp. 1–27, Jun. 2025, doi: 10.1145/3719006.
[2] M. Perkusich *et al.*, "Intelligent software engineering in the context of agile software development: A systematic literature review," *Information and Software Technology*, vol. 119, p. 106241, Mar. 2020, doi: 10.1016/j.infsof.2019.106241.
[3] K. T. Al-Sarayreh, K. Meridji, and A. Abran, "Software engineering principles: A systematic mapping study and a quantitative literature review," *Engineering Science and Technology, an International Journal*, vol. 24, no. 3, pp. 768–781, Jun. 2021, doi: 10.1016/j.jestch.2020.11.005.
[4] Moh. Z. N. Maulana and D. Siahaan, "Use Case-Based Analytical Hierarchy Process Method for Software Requirements Prioritization," in *2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, Yogyakarta, Indonesia: IEEE, Dec. 2022, pp. 205–210. doi: 10.1109/ICITISEE57756.2022.10057944.
[5] Moh. Z. N. Maulana, D. Siahaan, A. Saikhu, and E. Triandini, "Optimization Algorithm for Prioritizing Software Requirements: A Comparative Study," in *2024 7th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, Yogyakarta, Indonesia: IEEE, Dec. 2024, pp. 284–289. doi: 10.1109/ISRITI64779.2024.10963649.
[6] R. Asyrofi, D. O. Siahaan, and Y. Priyadi, "Extraction Dependency Based on Evolutionary Requirement Using Natural Language Processing," in *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, Yogyakarta, Indonesia: IEEE, Dec. 2020, pp. 332–337. doi: 10.1109/ISRITI51436.2020.9315489.
[7] F. Muftie and M. Haris, "IndoBERT Based Data Augmentation for Indonesian Text Classification," in *2023 International Conference on Information Technology Research and Innovation (ICITRI)*, Jakarta, Indonesia: IEEE, Aug. 2023, pp. 128–132. doi: 10.1109/ICITRI59340.2023.10250061.
[8] M. S. Haris, T. A. Kurniawan, and F. Ramdani, "Automated Features Extraction from Software Requirements Specification (SRS) Documents as The Basis of Software Product Line (SPL) Engineering," *JITeCS*, vol. 5, no. 3, pp. 279–292, Dec. 2020, doi: 10.25126/jitecs.202053219.
[9] I. K. Raharjana, D. Siahaan, and C. Fatichah, "User Story Extraction from Natural Language for Requirements Elicitation: Identify Software-Related Information from Online News," *SSRN Journal*, 2022, doi: 10.2139/ssrn.4297519.
[10] M. Z. N. Maulana, D. O. Siahaan, A. Saikhu, and E. Triandini, "Indonesian Software Description." figshare, p. 140581 Bytes, 2025. doi: 10.6084/M9.FIGSHARE.30343969.
[11] K. Kurniawan and A. F. Aji, "Toward a Standardized and More Accurate Indonesian Part-of-Speech Tagging," in *2018 International Conference on Asian Language Processing (IALP)*, Bandung, Indonesia: IEEE, Nov. 2018, pp. 303–307. doi: 10.1109/IALP.2018.8629236.
[12] M. Zhang, Z. Li, G. Fu, and M. Zhang, "Dependency-based syntax-aware word representations," *Artificial Intelligence*, vol. 292, p. 103427, Mar. 2021, doi: 10.1016/j.artint.2020.103427.
[13] S. Jiang, Z. Li, H. Zhao, and W. Ding, "Entity-Relation Extraction as Full Shallow Semantic Dependency Parsing," *IEEE/ACM Trans. Audio Speech Lang. Process.*, vol. 32, pp. 1088–1099, 2024, doi: 10.1109/TASLP.2024.3350905.
[14] J. W. Lim *et al.*, "Test case information extraction from requirements specifications using NLP-based unified boilerplate approach," *Journal of Systems and Software*, vol. 211, p. 112005, May 2024, doi: 10.1016/j.jss.2024.112005.
[15] I. K. Raharjana, D. Siahaan, and C. Fatichah, "User Stories and Natural Language Processing: A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 53811–53826, 2021, doi: 10.1109/ACCESS.2021.3070606.
[16] N. Ngaliah, D. Siahaan, and I. K. Raharjana, "User Story Extraction from Online News with FeatureBased and Maximum Entropy Method for Software Requirements Elicitation," *JTS*, vol. 32, no. 3, p. 125, Jan. 2022, doi: 10.12962/j20882033.v32i3.11625.
[17] Y. A. Kustiawan and T. Y. Lim, "User Stories in Requirements Elicitation: A Systematic Literature Review," in *2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS)*, Penang, Malaysia: IEEE, Aug. 2023, pp. 211–216. doi: 10.1109/ICSECS58457.2023.10256364.
[18] S. Sumesh and A. Krishna, "Hybrid analytic hierarchy process-based quantitative satisfaction propagation in goal-oriented requirements engineering through sensitivity analysis," *MGS*, vol. 16, no. 4, pp. 433–462, Dec. 2020, doi: 10.3233/MGS-200339.
[19] J. Hassine, D. Kroumi, and D. Amyot, "A Game-theoretic approach to analyze interacting actors in GRL goal models," *Requirements Eng*, vol. 26, no. 3, pp. 399–422, Sep. 2021, doi: 10.1007/s00766-021-00349-1.
[20] S. Sumesh and A. Krishna, "Challenges and review of goal-oriented requirements engineering based competitive non-functional requirements analysis," *MGS*, vol. 18, no. 2, pp. 171–191, Aug. 2022, doi: 10.3233/MGS-220231.
[21] S. Sumesh and A. Krishna, "Multi-agent based coalition formation of prosumers in microgrids using the i* goal modelling," *KES*, vol. 27, no. 1, pp. 25–54, Jul. 2023, doi: 10.3233/KES-230902.
[22] S. AlHajHassan, S. Murrar, A. Samhan, M. Odeh, and A. Elrashidi, "A Comparative Analysis of Goal-Oriented Requirements Engineering and Model-Based Systems Engineering Frameworks for Managing Requirements of Systems of Systems," in *2024 25th International Arab Conference on Information Technology (ACIT)*, Zarqa, Jordan: IEEE, Dec. 2024, pp. 1–12. doi: 10.1109/ACIT62805.2024.10877034.
[23] G. Pocevičė, P. Stefanovič, S. Ramanauskaitė, and E. Pavlov, "Approach for Tattoo Detection and Identification Based on YOLOv5 and Similarity Distance," *Applied Sciences*, vol. 14, no. 13, p. 5576, Jun. 2024, doi: 10.3390/app14135576.
[24] V. Nuipian and J. Chuaykhun, "Book Recommendation System based on Course Descriptions using Cosine Similarity," in *Proceedings of the 2023 7th International Conference on Natural Language Processing and Information Retrieval*, Seoul Republic of Korea: ACM, Dec. 2023, pp. 273–277. doi: 10.1145/3639233.3639335.
[25] J. Seo, H. Ko, and S. Park, "Space Authentication in the Metaverse: A Blockchain-Based User-Centric Approach," *IEEE Access*, vol. 12, pp. 18703–18713, 2024, doi: 10.1109/ACCESS.2024.3357938.