An Approach to Cluster Scenarios According to their Similarity using Natural Language Processing

Juliana Delle Ville[1], Diego Torres [1], Alejandro Fernández [1], Leandro Antonelli[1]

[1] Lifia, Fac. de Informática, UNLP, La Plata, Bs As, Argentina

{juliana.delleville, diego.torres, alejandro.fernandez, leandro.antonelli}@ lifia.info.unlp.edu.ar

Abstract. Scenarios are ideal to capture knowledge in human computer interface software engineering. Requirements engineering is a fundamental part of software development. If errors appear in this stage, it will be expensive to correct them in further stages. The domain experts and the developer team belong to different worlds. This generates a gap in communication between them. Because of it, it is important to use artifacts in natural language to communicate both sides. One simpler approach to specify requirements is Scenarios. They are widely used artifacts that generally describe the dynamics (tasks, activities) to be carried out in some specific situation. Generally, scenarios promote communication and participation from both sides. This can cause some problems. One of these problems is redundancy, that occurs when two stakeholders describe the same situation in different artifacts. This paper proposes an approach to analyze a set of scenarios by grouping them according to their similarity. The similarity is calculated through a series of comparisons of the different attributes of the scenario. This paper also describes a prototype implementing this method. Finally, the paper shows the result of a preliminary evaluation with results about the applicability of the approach.

Keywords: Scenarios, Natural language processing, Similarity.

1 Introduction

The scenarios are ideal to capture knowledge from the experts in Software Engineering in general, and they are even more necessary in human computer interface software engineering (HCI-SE)[3]. Requirements engineering is a critical stage of software development. Errors made at this stage can cost up to 200 times to repair when the software is delivered to the client [4].

Experts and development teams belong to different worlds and use different languages [16]. The experts use the language of the domain, while development teams use a computer science language. To cope with this communication gap, it is important to use artifacts in natural language that are readable by both parties [12].

Scenarios are widely used artifacts. Although scenarios have many conceptions, they generally describe the dynamics (activities, tasks) to be carried out in some specific situation, which should be different from the situation and dynamics described in other scenarios. Nevertheless, multiple scenarios might still depict the same objective. This definition applies for scenarios in software engineering as well as in finance, catastrophic events, etc. [7].

Scenarios are suitable to capture knowledge because they simply tell a story, and people know how to tell stories (funny anecdotes, stories for children, etc.). This story telling approach is effective because it is a way to incorporate details that are essential to provide a rich consolidation of knowledge. Because scenarios use natural language, experts can use them without the need to learn complex formalisms. Moreover, scenarios also promote communication and cooperation when there is a wide variety of experts [5], as many of them can describe different scenarios, improve them (if necessary), while learning from each other.

This collaborative writing of scenarios also introduces some challenges. For example, two stakeholders can describe the same situation in different artifacts (scenarios) with different levels of detail. Thus, redundancy appears. It is very important to provide tools to identify similar artifacts to avoid redundancy. Similar scenarios can be merged if they describe the same situation. If they are similar but describe different tasks to obtain the same result, they need to be enriched to express the details that make them different.

If we consider texts as sets of words, we can compare them using Jaccard's similarity index [29]. Jaccard's similarity is defined as the size of intersection of the sets divided by the size of the union of the sets. The higher the value, the more similar they are.

Nevertheless, comparing two scenarios is not as easy as directly computing Jaccard's similarity index on their texts. The scenario has a structure, and this structure can be used to assess similarity with better results than simply applying Jaccard's similarity (or any other method) directly to the whole description of the scenario.

This paper proposes a method to assess similarity among a group of scenarios. Particularly, the method consists in comparing scenarios by pairs, and according to their similarities, the groups of scenarios are ranked. Thus, this method can be seen as a way of ranking pairs of scenarios by similarity so an expert can deal with them and finally decide whether scenarios are similar or not.

The rest of the paper is organized in the following way. Section 2 presents background concepts about scenarios. Section 3 details our contribution, that is, the proposed approach. Section 4 describes the tool to support the proposal. Section 5 discusses related work. Finally, Section 6 discusses some conclusions.

2 Scenarios

A scenario is an artifact that describes situations in a specific domain using natural Language [5]. It describes a situation that occurs in a specific context to reach a cer-

tain goal. There is a sequence of steps to achieve that goal: the scenario's episodes. These episodes describe actions that are performed by actors using resources.

Scenarios can be seen as descriptions of real-world situations, captured in a set of small stories [8][6]. Then, these scenarios can be used in more complex artifacts to model software requirements. For example, one Use Case can include several Scenarios: a scenario describing the happy path, another scenario that describes the alternative path, and another scenario that describes the exceptional path [1][28].

There are many proposals to represent scenarios. Leite [10] proposes a structure with the following attributes: a title, a goal, a context, the actors, the resources, and a list of episodes. The goal of a scenario is the objective to be attained by executing the scenario. The actor is the subject who performs the actions described in the episodes. The context is defined by the place, time, and conditions that allow the scenario to start. The resources are the tools, materials, and data necessary to perform the scenario. And finally, the episodes are a collection of tasks described using an actor, an action, and a resource.

Let's consider the agricultural domain where a farmer sow's tomatoes. In that domain, the irrigation can be done manually (with a watering can), or it can be done with some infrastructure such as pipes and a pump. The scenario "Irrigate by hand" (Table 1) and "irrigate manually" (Table 2) describe through different scenarios the same activity performed using a watering can. Then, the scenario "irrigate with pump" (Table 3) describes the irrigation using a complex infrastructure of pipes and valves. Finally, the scenario "sow tomato seeds" (Table 4) describes how to put the seed into the soil.

Table 1. Description of the scenario "Irrigate by hand".

Attributes	Description
Scenario	Irrigate by hand.
Goal	Provide H2O to the tomato.
Context	The tomato plant is in any state of grow.
Resources	Water, watering can.
Actors	Farmer.
Episodes	The farmer fills the watering can with water. The farmer pours the water to the base of the plant.

3 The proposed approach

3.1 Our approach in a nutshell

The proposed approach aims to analyze a group of scenarios, comparing them pairwise and sorting the pairs based on their similarity. Then, an expert can analyze the pairs of scenarios (from the most to the least similar) to confirm whether the scenarios are the same (in this case, they should be merged) or not (in this case their differences should be emphasized). The similarity of the scenarios is calculated by

comparing their attributes: title, goal, context, actors, resources, and episodes. The approach is summarized in the algorithm depicted in Table 5.

Table 2. Description of the scenario "Irrigate manually".

Attribute	Description
Scenario	Irrigate manually.
Goal	Supply water to the tomato plant.
Context	Have the tomato plant in any state of grow, but mainly in the fruit formation stage.
Resources	Water, watering can, worm leachate.
Actors	Farmer.
Episodes	The farmer fills the watering can with water. The farmer adds worm leachate to the watering can. The farmer approaches the tomato plant. The farmer visually assesses the humidity of the soil. The farmer approaches the watering can to the plant's base. The farmer pours the water.

Table 3. Description of the scenario "Irrigate with pump".

Attribute	Description
Scenario	Irrigate with pump.
Goal	Provide water for the seeds and the tomato plants.
Context	Tomato plant in any state of grow. Deployed an infrastructure of pipes, valves, pumps.
Resources	Cistern with enough water.
Actors	Farmer, technician.
Episodes	The farmer determines the sector to irrigate. The technician opens the valves of the sector to irrigate. The farmer decides the intensity of the irrigation. The technician sets the pump to the intensity of the irrigation. The technician starts the pump.

Table 4. Description of the scenario "Sow tomato seeds".

Attribute	Description
Scenario	Sow Tomato Seeds.
Goal	Place the Tomato Seeds in the seedbed.
Context	The seedbeds are already prepared.
Resources	Seed, substrate, water.
Actors	Farmer.
Episodes	The farmer digs a hole in the seedbed. The Farmer places the seeds in the seedbed. The Farmer covers the seeds with substrate. The farmer sprays the seedbed with water.

Table 5. Algorithm of the approach.

Line	Code
1	Similarity (Si.attribute, Sj.attribute)-> si.attribute
2	Convert to lemma (Si.attribute) -> si.attribute
3	remove stop words (Sj.attribute)-> sj.attribute
4	Convert to lemma (Sj.attribute) -> sj.attribute
8	return Jaccard_Similarity (Si.attribute, Sj.attribute)
9	Jaccard_Similarity (Si.attribute, Sj.attribute)
10	Local intersection
11	Local union
12	calculate intersectionof(Si.attribute,Sj.attribute) ->intersection
13	calculate union of (Si.attribute, Sj.attribute)-> union
14	Return size(intersection) / size(union)
15	rank (scenarios)
16	for each scenario Si, Sj in scenarios
17	Similarity (Si.title, Sj.title)-> title.similarity
18	Similarity (Si.goal, Sj.goal)-> goal.similarity
19	Similarity (Si.context, Sj.context)-> context.similarity
20	Similarity (Si.resources, Sj.resources) -> resources.similarity
21	Similarity (Si.actors, Sj.actors) -> actors.similarity
22	Similarity (Si.episodes, Sj.episodes) -> episodes.similarity
23	Title.similarity + ((goal.similarity + context.similarity)/2) + ((actors.similarity + resources.similarity)/2) + episodes.similarity)/4->rank
24	Answer add Pair (Si, Sj) with rank rank
25	Return answer order by rank descending.

The approach consists of comparing attributes by attributes to obtain a general assessment of similarity. Concepts and verbs are extracted from every attribute and a relation between number of equal elements (the intersection) divided the total number of elements (the union) is obtained (this is Jaccard's method). Then, the similarity between two scenarios is calculated. To do so, some attributes are grouped (goal with context, and actors with resources).

It is important to mention that the comparison is merely syntactical, that is, synonyms are considered as different words. For example, provide and supply, H2O and water, tomato plant and plant, base of the plant and plant's base are different concepts for the approach.

The steps to compare the expressions are the following. First, every word is converted to its root form (lemma). And then, the words with semantic meaning are filtered (that is, the stop words are removed).

3.2 Example

Let's consider the four scenarios that describe situations in the agricultural domain depicted in Tables 1, 2, 3, and 4. Two scenarios are quite similar and describe how to

irrigate manually (let's identify them as Scenario A, the one described in Table 1, and Scenario B, the one described in Table 2). Then, another scenario describes how to irrigate with pipes and pumps (Table 3, identify it as Scenario C), so it resembles the previous scenario, but it is not so similar. Then, the fourth scenario describes how to sow seeds (Table 4, let's identify it as Scenario D). This scenario is quite different from the previous ones.

Scenario A and B are quite similar since both describe how to irrigate manually. The differences between the scenarios are two. Firstly, scenario A is shorter than scenario B, since scenario A only describes how to supply water while scenario B also considers worm leachate. Scenario B also describes in detail how to pour the water. Secondly, scenarios A and B use synonyms or different expressions to refer to the same elements or actions. The following table 6 and 7 summarizes the comparison and its final rank which is 0.635

Table 6. Comparison between scenario A and B (attribute by attribute).

	Scenario A	**Scenario B**	**Intersection / Union**
Title	Irrigate	Irrigate	1 / 1 = 1
Actors	Farmer	Farmer	1 / 1 = 1
Resources	Water, watering can	Water, watering can, worm leachate	2 / 3 = 0.66
Goal	Provide, h2o, tomato	Supply, water, tomato plant	0 / 6 = 0
Context	Tomato plant, state of grow	Tomato plant, state of grow, fruit formation stage	2 / 3 = 0.66
Episodes	Farmer, fill, watering can, water, pour, base of the plant	Farmer, fill, watering can, water, add, worm leachate, approach, tomato plant, assess, humidity of the soil, plant's base, pour	5 / 13 = 0.38

Table 7. Comparison between scenario A and B final rank.

Attributes	**Similarity**
Title	1
Actors and resources	(1 + 0.66) / 2 = 0.83
Goal and context	(0 + 0.66) / 2 = 0.33
Episodes	0.38
Final rank (average)	2.54 / 4 = 0.635

Scenario A and C describe the same activity (irrigation), but it is done in different ways. Scenario A describes how to irrigate manually, while scenario C describes how to irrigate with a pump (a complex infrastructure of pipes, valves and pumps). The following table 8 and 9 summarizes the comparison. The final rank is 0.305.

Table 8. Comparison between scenario A and C (attribute by attribute).

	Scenario A	Scenario C	Intersection / Union
Title	Irrigate	Irrigate, pump	1 / 2 = 0.5
Actors	Gardener	Farmer, technician	1 / 2 = 0.5
Resources	Water, watering can	Cistern, water	1 / 3 = 0.33
Goal	Provide, h2o, tomato	Provide, water, seed, tomato plant	1 / 6 = 0.16
Context	Tomato plant, state of grow	Tomato plant, state of grow, deploy, pipe, valve, pump	2 / 6 = 0.33
Episodes	Farmer, fill, watering can, water, pour, base of the plant	Farmer, determine, sector to irrigate, technician, open, valve, decides, intensity of the irrigation, set, pump, start	1 / 15 = 0.06

Table 9. Comparison between scenario A and C final rank.

Attributes	Similarity
Title	0.5
Actors and resources	(0.5 + 0.33) / 2 = 0.415
Goal and context	(0.16 + 0.33) / 2 = 0.245
Episodes	0.06
Final rank (average)	1.22 / 4 = 0.305

Scenario A and D describe completely different activities: irrigation and sowing. The following table 10 and 11 summarizes the comparison. The final rank is 0.191.

Table 10. Comparison between scenario A and D (attribute by attribute).

Scenario		Scenario D	Intersection / Union
Title	Irrigate	Sow, tomato seed	0 / 3 = 0
Actors	Farmer	Farmer	1 / 1 = 1
Resources	Water, watering can	Seed, substrate, water	1 / 4 = 0.25
Goal	Provide, h2o, tomato	Place, tomato seeds, seedbed,	0 / 6 = 0
Context	Tomato plant, state of grow	Seedbed, prepared	0 / 4 = 0
Episodes	Farmer, fill, watering can, water, pour, base of the plant	Farmer, dig, hole, seedbed, places, seeds, covers, substrate, spray, water	2 / 14 = 0.14

Table 11. Comparison between scenario A and D final rank.

Attributes	Similarity
Title	0
Actors and resources	(1 + 0.25) / 2 = 0.625
Goal and context	(0 + 0) / 2 = 0
Episodes	0.14
Final rank (average)	0.765 / 4 = 0.191

Table 12 summarizes the comparison among all the scenarios.

Table 12. Final rank of the scenarios

Scenario i	Scenario j	Rank
Scenario A	Scenario B	0.635
Scenario A	Scenario C	0.305
Scenario B	Scenario C	0.230
Scenario A	Scenario D	0.191
Scenario B	Scenario D	0.155
Scenario C	Scenario D	0.010

4 Tool support

A software tool was prototyped to assist the application of the proposed method. The prototype is a web application written in Python [18] using Spacy [25] and NLTK [15] libraries to deal with natural language processing.

As input, the prototype receives a set of scenarios and as output it produces a set of tuples with the following information ('rank', 'scenario_i', 'scenario_j'). The application processes every possible pair of scenarios and calculates the rank of similarity for every pair. Figure 1 describes a snapshot of the application.

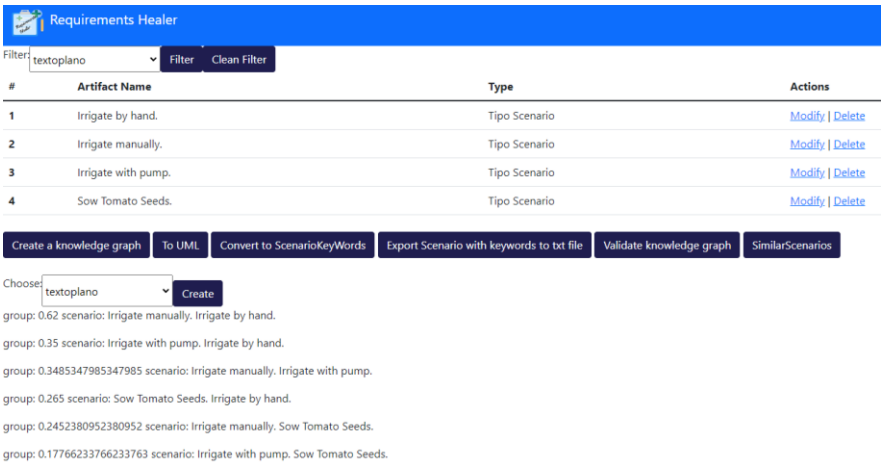

Figure 1. Prototype with the scenarios used as example.

The process to calculate the similarity has different steps. The first step consists in tokenizing the scenarios. It can be done in two different ways. If the attribute is a list of words (actors and resources), it is processed by lowering them and removing stop words. If the attribute is a sentence or a group of sentences (title, goal, context, episodes), stopwords are removed and they are lemmatized and lowered. Then, nouns and verbs are collected. Then, Jaccard's method receives the data from the two scenarios from the previous step and calculates the jaccard similarity. This result and the scenarios are stored in a tuple as ('rank', 'scenario_i', 'scenario_j'). Figure 2 summarizes this process.

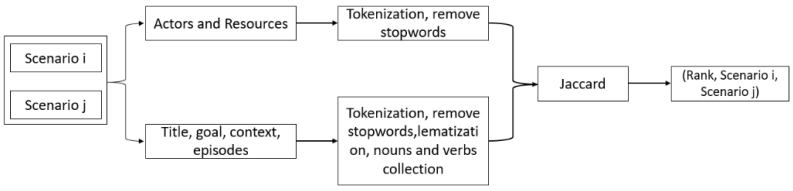

Figure 2. Summary of the implementation.

5 Related works

There are many works that use Jaccard to determine similarity in natural language artifacts. Some of them analyze requirements specification [14], some other are used in information retrieval [22], while some others are used for bugs reports [23][9][27]. Instead of Jaccard's similarity, other works [19] [13] rely on cosine similarity. And some other defines their own method [30][2][20][9].

Regarding requirements specification, some papers analyze User Stories [2] to search duplicated or to compare similarity with the use case descriptions [17], some others analyze UML diagrams [30] to measure similarity between use case description and sequence diagram in a requirement specification, and some others uses documents without an specific template to evaluate the benefits of automatic similarity analysis [14]. There is one approach [30] that analyzes UML (Uses Cases) and documents at the same time. Another approach is by using similarity in queries [22] to obtain better results in the search. And another approach is by using a discriminating model [26] detect duplicated requirements.

Many methods rely on syntactic similarity [24][11], although some of them also relies on semantic similarity [2] [19] [20].

Priyadi et al. [17] propose a method to assess the similarity of User Stories but they do not focus simply on the artifact, they also deal with the elicitation process to determine the suitability between the requirements elicitation and requirement modeling. Barbosa et al. [2] propose a method to assess the similarity of User Stories in the scrum process based on Jaccard and cosine similarity for syntactic similarity. The method proposed in this paper only focuses on the requirement artifact scenario and can be used in any elicitation technique.

Yanis et al. [30] deal with UML, particularly with Use Cases and Sequence diagrams studying the similarity between them and the section object of a software development document to determine the suitability of these two. In this paper, the method proposed only focuses on text similarity. Sari et al. [24] propose a study of semantic similarity via Wu Palmer method through functional requirements with use case diagrams. al. [13] propose a method to obtain a linkage between software reusability and similarity text, to find similarity between projects and reuse components. The method proposed in this paper uses the Jaccard similarity method to cluster scenarios by syntactic similarity through its field. Dag et al. [14] analyzed the benefits of automated similarity analysis of textual requirements focused on market-driven development. This paper is not focused on any development method, is an approach with a general application of scenarios to help to determine duplicated scenarios. Rao et al. [21] propose a method to detect duplicate requests in the requirements analysis stage using similarity techniques. The approach proposed in this paper focuses on the written scenarios not on the requests. Rago et al. [20] propose a tool, reqAligner, that combines text processing techniques and creates an abstract representation to identify duplicated functionality with semantic similarity. This paper only focuses on syntactic similarity.

6 Conclusions and future work

This paper described a method to rank a set of scenarios according to their similarity. This method compares every pair of scenarios to scenarios to determine the level of similarity between them, and then it sorts them in order from the most to the least similar. This paper also presents a software prototype implementing the proposed method. The results are promising, even though there is still much work to be done. The work of different people in the same project obtaining scenarios can generate a duplication of different scenarios with different levels of details. The contribution of the proposed approach can help detecting similar scenarios. Nevertheless, this approach relies on syntactic similarity. This analysis can be more complex with semantic similarity. Because different people can describe the same activity or object in different words. For example, watering the plant is the same as irrigating the plant. This can be even more complex with hypernyms. Thus, we plan to continue with the proposal adding some semantic similarities. Moreover, it is necessary to perform a detailed validation of the proposed approach. That is, we plan to perform a case study in order to assess the usability and applicability of the proposed approach. And we also plan to perform an experiment in order to assess the effectiveness of the proposed approach in comparison with some other approaches. Finally, we plan to improve the prototype with some usability functionality. Particularly, we believe that awareness is an important feature that should be included to foster practitioners to use the tool (hence the approach).

References

1. Alexander, I., Maiden, N.: Scenarios, Stories, Use Cases, through the system development life cycle, John Wiley & Sons (2004).
2. Barbosa, R., Silva, A. E. A., Moraes R.: Use of similarity measure to suggest the existence of duplicate user stories in the scrum process. In Proc 46th annual IEEE/ifip international conference on dependable systems and networks workshop. IEEE (2016).
3. Benyon, D., Macaulay, C.: Scenarios and the HCI-SE design problem. Interacting with Computers, 14(4), 397–405. doi:10.1016/s0953-5438(02)00007-3 (2002).
4. Boehm, B.W.: Software Engineering, Computer society Press, IEEE, (1997).
5. Carrol, J. M.: Five reasons for scenario-based design, in Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences pp. 2-5. (1999).
6. Carroll J.M.: Making Use: Scenario-Based Design of Human-Computer Interactions, Cambridge MA, MIT Press (2000).
7. Cockburn A.: Writing Effective Use Cases, Addison-Wesley, Boston MA (2001).
8. Gough P.A., Fodemski F.T., Higgins S.A., Ray S.J.: Scenarios: An Industrial Case Study and Hypermedia Enhancements, in Proc IEEE International Symposium on Requirements Engineering (RE '95), IEEE Computer Society Press, Los Alamitos CA, pp. 10-17, (1995).
9. Khtira, A., Benlarabi, A., El Asri, B.: Detecting feature duplication in natural language specifications when evolving software product lines. In Proc International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). IEEE, pp. 257-262. (2015).

10. Leite, J. C. S. d. P., Rossi, G., Balaguer, F., Maiorana, V., Kaplan, G., Hadad, G. Oliveros, A.: Enhancing requirements baseline with scenarios, Requirements Engineering Journal, vol. 2, no. 4, pp. 184-198 (1997).
11. Lerch, J., Mira M.: Finding duplicates of your yet unwritten bug report. 17th European conference on software maintenance and reengineering, IEEE, pp. 69-78. (2013).
12. Lim, S. L., Finkelstein, A.: StakeRare: Using Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation, IEEE transactions on software engineering, Volume 38, Issue 3, May-Jun 707-735, (2012). doi 10.1109/TSE.2011.36
13. Mihany, F. A., Moussa, H., Kamel, A., Ezzat, E., Ilyas, M.: An automated system for measuring similarity between software requirements. Proceedings of the 2nd Africa and Middle East Conference on software engineering (2016).
14. Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., Karlsson, J.: A feasibility study of automated natural language requirements analysis in market-driven development, Requirements engineering Journal, vol 7, pp 20-33 (2002).
15. NLTK Natural Language Toolkit, https://www.nltk.org/, accessed 17/5/2023.
16. Potts, C.: Using schematic scenarios to understand user needs, in Proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques, pp. 247-256. (1995)
17. Priyadi, Y., Putra, A. M., Lyanda, P. S.: The similarity of Elicitation Software Requirements Specification in Student Learning Applications of SMKN7 Baleendah Based on Use Case Diagrams Using Text Mining. In proc 5th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE). IEEE, pp. 115-120. (2021).
18. Python, https://www.python.org/, accessed 17/5/2023.
19. Qurashi, A. W., Holmes, V., Johnson, A. P.: Document processing: Methods for semantic text similarity analysis. In Proc International Conference on INnovations in Intelligent SysTems and Applications (INISTA), IEEE, pp. 1-6. (2020).
20. Rago, A., Marcos, C., Diaz-Pace, J. A.: Identifying duplicate functionality in textual use cases by aligning semantic actions. Software & Systems Modeling 15.2, pp. 579-603. (2016).
21. Rao, D., Bian, L., Zhao, H.: Research of Duplicate Requirement Detection Method. Smart Computing and Communication. In Proc 7th International Conference, SmartCom 2022, New York City, NY, USA, November 18–20, Springer Nature Switzerland, pp. 213-225. (2023).
22. Rinartha, K., Suryasa, W.: Comparative study for better result on query suggestion of article searching with MySQL pattern matching and Jaccard similarity. In Proc 5th International Conference on Cyber and IT Service Management (CITSM), IEEE, pp. 1-4. (2017).
23. Runeson, P., Alexandersson, M., Nyholm, O.: Detection of duplicate defect reports using natural language processing. In Proc 29th International Conference on Software Engineering (ICSE'07), IEEE, pp. 499-510. (2007).
24. Sari, E. J., Priyadi, Y., Riskiana, R. R.: Implementation of Semantic Textual Similarity Between Requirement Specification and Use Case Description Using WUP Method (Case Study: Sipjabs Application). In Proc IEEE World AI IoT Congress (AIIoT), IEEE, pp. 681-687. (2022).
25. spaCy · Industrial-strength Natural Language Processing in Python, https://spacy.io/, accessed 17/5/2023.
26. Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S. C.: A discriminative model approach for accurate duplicate bug report retrieval. In Proc Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, vol. 1. pp. 45-54. (2010).

27. Sureka, A., Jalote, P.: Detecting duplicate bug report using character n-gram-based features. In Proc Asia Pacific software engineering conference, IEEE, pp. 366-374. (2010).
28. Sutcliffe, A. G., Maiden, N. A., Minocha, S., Manuel, D.: Supporting Scenario-Based Requirements Engineering, IEEE Transactions on Software Engineering, vol. 24, pp 1072-1088. (1998).
29. vor der Brück, T., Pouly, M.: Text similarity estimation based on word embeddings and matrix norms for targeted marketing. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies vol 1 pp. 1827-1836. (2019).
30. Yanis, R. Z. I., Priyadi, Y., & Puspitasari, S. Y.: Measurement of Similarity between Use Case Description and Sequence Diagram in Software Requirement Specification using Text Analysis for Dtrain Application. In Proc 2nd International Conference on Electronic and Electrical Engineering and Intelligent System (ICE3IS), IEEE, pp. 328-333. (2022).