A Smart AI Framework for Backlog Refinement and UML Diagram Generation

Samia NASIRI*, Mohammed LAHMER
Moulay Ismail University, Faculty of Science, Meknes, Morocco

***Abstract*—In Agile development, it is crucial to refine the backlog to prioritize tasks, resolve problems quickly, and align development efforts with project goals. Automated tools can help in this process by generating Unified Modeling Language (UML) diagrams, allowing teams to work more efficiently with a clear understanding and communicate product requirements. This paper presents an automated approach to Agile methodology which refines backlogs by detecting duplicate user stories and clustering them. Following the refinement process, our approach generates UML diagrams automatically for each cluster, including both class and use case diagrams. Our method is based on machine learning and natural language processing techniques. To implement our approach, we developed a tool that selects the user stories file, groups them by actor, and employs the unsupervised k-means algorithm to form clusters. After that, we used Sentence Bidirectional Encoder Representations from Transformers (SBERT) to measure the similarity between user stories in a cluster. The tool highlights the most similar user stories and facilitates the decision to delete or keep them. Additionally, our approach detects similar or duplicate use cases in the UML use case diagram, making it more convenient for computer system designers. We evaluated our approach on a set of case studies using different performance measures. The results demonstrated its effectiveness in detecting duplicate user stories in the backlog and duplicate use cases. Our automated approach not only saves time and reduces errors, but it also improves collaboration between team members. With an automatic generation of UML diagrams from user stories, all team members can understand product requirements clearly and consistently, regardless of their technical expertise.**

Keywords*—Artificial intelligence; NLP; Agile methodology; UML

I. Introduction

In the System Development Life Cycle, it is essential to undergo a requirement analysis stage to ensure the success of the process [1]. This occurs because developers must understand the requirements before proceeding to the implementation stage. In this context, user stories are increasingly used to communicate requirements in Scrum. They are semi-structured natural language expressions of requirements at a high level. The textual template for user stories has been proposed in many forms by practitioners. In practice, they tend to use the form of: “As a…, I want to…, so that…” The growing number of stakeholders involved in the development process increases the size of the systems developed, leading to a larger number of user stories. This size growth makes it mandatory to decompose the system into subsystems covering different sets of semantically similar user stories.

As part of requirements engineering tasks, stakeholders need to be kept involved in the process by expressing each sub-system semi-formally, such as using visual models. As a visual language supporting requirements engineering, we used the Unified Modeling Language (UML) in this context. A model-based approach is often used to specify software system features and to reduce ambiguity between requirement specification and design. However, deriving UML models manually from similar user stories can be time-consuming and tedious, especially for large systems. Additionally, model-based approaches have been difficult to integrate in Scrum processes. This is basically due to the lack of powerful automation tools [2], [3], as well as focusing on implementation rather than analysis and documentation of teams.

Several studies were conducted to automate the generation of models from natural language software requirements [4]-[12]. In addition, some authors studied natural language requirements clustering to decompose the target system at an initial level [13], [14]. The current requirements clustering approach lacks precision and does not achieve a high level of automation. In contrast, we propose a machine learning-based approach that addresses these challenges. However, the requirement clustering has rarely been considered as a primer for automatically deriving models.

In this paper, we propose a machine learning-based approach for automatically dividing a system into subsystems and generating UML diagrams based on natural language user stories in Scrum.

To group the user stories by actor, we applied a set of natural language processing heuristics to extract the actor name from each user story. Once the system was initially decomposed by the actor, we performed a second decomposition for each resulting cluster. The second clustering of the system was based on the k-means algorithm, which groups semantically similar requirements. To identify possible redundancies between user stories located in each cluster, we used the BERT model. This process allows Product Owners and Scrum Masters to be more informed about their decisions regarding the elimination of redundant user stories from the Product Backlog. In the final step, we generated use case and class UML diagrams from the identified clusters.

This paper is organized as follows. The Section II reviews related work, while the Section III presents the background of the proposed approach. Section IV is devoted to a detailed description of the proposed approach. In Section V, we describe and analyze the similarity detection between user stories and generated UML diagrams; it also evaluates and discusses the performance of the approach. Finally, Section VI concludes the paper.

II. Related Work

In this section, we provide a literature review related to our approach. For instance, in study [15], the authors have reviewed word embeddings and implemented a convolutional neural network trained on pre-trained word vectors. They have also illustrated the performance of pre-trained word integrations vs. random embeddings. However, in study [16], the authors have presented an approach that uses semantic similarity measures to suggest possible cases of duplication between user stories. To explain, the approach selected the most appropriate measure to determine the level of similarity between user stories. Their research analyzed semantic similarity measures based on the WordNet lexical database WuP, a similarity measure based on VSM Lin [17], and a measure based on the frequency of common terms Lesk-A [18]. The authors of study [19] analyzed user stories to identify potential information gaps and prevent ambiguities, using comparisons with previous user stories to detect missing queries. They have provided suggestions to users to get a better description. For natural language processing (NLP), an NLP tool called LingPipe Toolkit is used. Semantic role labeling was carried out to attribute roles and actions. In study [20], the authors' method established the user stories meta-model by determining the unified descriptive model of the user stories which are: the role, the task, the capability, the soft goal, and the hard goal. In study [21], the approach allowed the extraction of relevant information for user stories from recorded conversations between customers and developers. In study [13], the authors' work consisted of clustering the specification requirements by first using the Vector Space Model (VSM) to compute the similarity of functional requirements and then the Agglomerative Hierarchical Clustering (AHC) algorithm to construct clusters. In study [22], the authors have proposed a tool-assisted approach to identify terminological ambiguities between viewpoints as well as missing requirements. For this purpose, they combined natural language processing with information visualization techniques that help in defect-type interpretation. Their approach consisted of identifying ambiguity and incompleteness in a set of requirements. The authors used word2vec to detect similarities between terms in requirements. The visualization showed the requirements graphically by marking the terms used and arranging them in a 2D space according to the viewpoint to which the terms belong. They used Cortical.io's algorithm which relies on semantic folding and fingerprinting. Then, the algorithm built the context of each pair of terms that appeared in the same user stories. In study [23], the authors provided Sentence-BERT (SBERT). This pre-trained BERT network modification takes advantage of Siamese and triplet network structures by deriving semantically meaningful sentence embeddings that can be benchmarked using cosine similarity. In study [24], the authors used Word2vec to compute the similarity at the word level, then they grouped the requirements into clusters using the Agglomerative Hierarchical Clustering (AHC) algorithm. Word2vec for each word after tokenization at remove stop words, and stem. They used Gensim API for keyword extraction of each cluster. This summarizer is based on the ranks of text sentences using a variation of the TextRank algorithm. After that, they defined simple NLP rules for component extraction to generate a use case diagram. In [14] authors used a K-means clustering algorithm applied to user stories. The authors in study [25] have provided a tool to extract the time spent in historical and similar user stories. This extracted time helps developers estimate the time that similar user stories will spend on new projects. To do this, the authors used the NLP algorithm and a pre-trained model developed by Google called USE. The model did not distinguish between the opposite operations “add and remove”. In research [26], the method began by extracting iStar nodes from semi-structured user stories. Next, the nodes were merged based on their similarity. Finally, edges between nodes were identified using defined rules. The authors' approach was based on the refinement relationship between iStar nodes. The authors applied a BERT (Bidirectional Encoder Representations from Transformers) model for similarity measurement. In study [27], the approach helped in detecting defects in user stories by using 11 quality criteria. It was based on two main components: firstly, quality analysis was based on NLP. This method uses four fundamental functions of natural language processing: sentence segmentation, tokenization, POS labeling, and syntactic analysis. It examined the completeness of components and the testability of stories by checking several quality criteria, such as the correct form of components and the consistency of keywords used in stories. Secondly, the method was based on the analysis of iStar models. It started by generating iStar models from user stories, identifying nodes (role, goal/task), and detecting relationships between these nodes. It then checked several quality criteria, including uniqueness, absence of conflicts, verifiability, independence, and conceptual consistency.

Many approaches for requirements gathering and analysis have been developed so far, falling into two main categories: (1) approaches based on gathering, and (2) approaches focused on identifying duplicates or ambiguities in user stories. Our approach combines these two methods while refining the user stories to eliminate duplicates. In addition, the proposed approach aims at automatically generating UML diagrams from a given set of refined user stories.

III. Background

In this section, we outline the key concepts underlying this work.

A. Text to Semantic Vectors Transformation in NLP

It is important to transform text into semantic vectors in many automatic NLP tasks. This enables algorithms to process language more effectively by representing the meaning of words and sentences. There are several approaches for converting text into semantic vectors, and each one of them has its strengths and limitations. Hence, in this section, we will discuss the most common approaches, including One Hot Encoding, TF-IDF, Word2Vec, ELMo, InferSent, and Sentence Transformers [28].

- One Hot Encoding is a straightforward method that transforms text into binary vectors by assigning a unique dimension to each word in the corpus. In this method, the dimension corresponding to each word is marked as 1 for each document, while all other dimensions are marked as 0. The resulting vectors are binary vectors of a size equal to the total number of words in the corpus. Although this method is simple to implement, it disregards semantic relationships between words and can result in large and sparse vectors.

- TF-IDF is a technique that represents documents as vectors using term frequency and inverse document frequency. The weight of a word in a document is computed by multiplying its frequency in the document (TF) by the inverse of its frequency in the entire corpus (IDF). This approach generates weighted vectors that consider the relative importance of words in documents. Even if it is more expressive than One Hot Encoding, TF-IDF still does not capture semantic relationships between words [29].

- Word2Vec is a neural network-based method that learns dense vector representations of words in a vector space. It includes two main variants: Skip-Gram and Continuous Bag of Words (CBOW). Skip-Gram predicts neighboring words given a target word while CBOW predicts the target word using its neighboring words. The resulting vectors capture semantic and syntactic relationships between words. However, Word2Vec does not consider the syntactic structure of sentences [30] [31].

- ELMo, or Embeddings from Language Models is a model that represents words and phrases in automatic natural language processing. It creates contextual embeddings using bidirectional recurrent neural networks (Bi-LSTMs), capturing the meaning of words in context. Pre-trained on unannotated texts, ELMo generates rich, contextual embeddings. These representations enhance text classification, information retrieval, and other NLP tasks by capturing the semantic and syntactic nuances of words in various contexts.

- InferSent, developed by Facebook AI Research (FAIR), is a sentence representation model that generates contextual semantic embeddings using deep neural networks. Based on a semi-supervised supervision approach, InferSent aims to capture the semantic meaning of sentences. Pre-trained on a large corpus of data, this model produces informative sentence embeddings, beneficial for tasks such as text classification and semantic similarity assessment between sentences.

- The Universal Sentence Encoder (USE) is a widely adopted natural language processing model that has been extensively used in various research domains. Developed by Google, USE employs a deep neural network to encode text into fixed-dimensional vectors, capturing semantic information and contextual meaning [32]. It has proven to be effective in tasks such as sentence similarity, document classification, and semantic search. With its ability to understand and represent the meaning of sentences, USE has become a valuable asset in numerous applications. Its success lies in its capability to capture intricate semantic relationships. This makes it a reliable tool for tasks involving text comprehension and similarity analysis.

- Sentence Transformers: The advancement of deep learning has resulted in a significant improvement in the performance of neural network architectures like recurrent neural networks (RNN and LSTM) and convolutional neural networks (CNN) in the areas of Natural Language Processing (NLP), such as text classification, language modeling, and machine translation.

Bidirectional Encoder Representations from Transformers is a Transformer-based machine learning technique for natural language processing: pre-training developed by Google. The Sentence BERT (SBERT) network uses siamese and triplet networks to obtain semantically meaningful sentence embeddings derived from BERT [23]. SBERT can be employed as both a semantic similarity search and as a clustering algorithm. Similarity measures like cosine similarity or Manhattan/Euclidean distance can be used to determine semantic similarity.

Several SBERT pre-trained models encode sentences and calculate the distance between them to conduct semantic searches. Each model has its own task such as question answering, translation, sentence similarity, and others. These models are tuned for many use cases and trained on a huge and varied dataset consisting of over a billion training pairs. They are called “sentence transformers” and represent a modern approach to transforming text into dense semantic vectors using pre-trained neural network models. One of the main advantages of sentence transformers lies in their ability to capture the full semantic meaning of entire sentences, beyond representations of individual words. Notably, these models are often trained for natural language understanding tasks, such as next-sentence prediction or text classification, to further enhance their performance and accuracy. On the one hand, these models consider the contextual relationships between words in a sentence and generate representations that capture the meaning of the sentence as a whole. On the other hand, word-based models only consider the individual words and their relationships to other words in the corpus. This allows sentence transformers to more effectively capture the meaning and semantic nuances of a sentence, which is particularly beneficial for tasks involving the evaluation of similarity between user stories.

B. *Grouping of user Stories by Actor*

Grouping textual requirements based on their semantic similarity provides valuable information on the structure and behavior of the target system. To explain, it simplifies interpretation and identifies redundancies and inconsistencies which improves system quality. In addition, it enables efficient requirements management, traceability, and impact analysis, improving the overall system design and development process.

In this context, we aim to group user stories according to their semantic similarity.

Unsupervised learning is a machine learning technique that works on unlabeled data. In this technique, the machine is not given predefined labels to use for learning, but autonomously identifies patterns in the data to solve the business problem.

In clustering, unlabeled data is assembled into groups by using an algorithm based on unsupervised learning. In each cleaned dataset, with the help of the algorithm, the data points can be organized into groups by using the clustering algorithm. As a result of the clustering algorithm, it is presumed that the data points that belong to the same group, will have similar properties whereas the data points, that belong to different groups, will have quite different properties. Machine learning algorithms include a wide variety of clustering algorithms. Among the various clustering algorithms used in machine learning, K-Means is the most commonly used. According to [33], the use of K-means offers significant advantages for clustering textual requirements. This algorithm is appreciated for its speed, simplicity, and interpretability.

In addition, it offers flexibility by allowing the desired number of clusters to be specified. This study confirmed the benefits of K-means in the efficient clustering of requirements.

The process of the K-Means algorithm is outlined in the steps below:

- Step 1: We first select a random number of K to use and randomly initialize their respective center points.
- Step 2: We then classify each data point by calculating the distance (Euclidean or Manhattan) between that point and the center of each cluster, and then regrouping the data point so that it is in the cluster whose center is closest to it.
- Step 3: We recalculate the cluster center by averaging all vectors in the cluster.
- Step 4: We repeat all these steps for n number of iterations or until we find that the cluster centers do not change much.

IV. Approach

This section presents our proposed approach to backlog refinement and UML diagram generation. This approach both identifies similar user stories and generates the corresponding UML diagrams for each group spotted. First, we give an overview of our proposal. Then, we provide a detailed explanation of each step.

Our method started by grouping user stories by actor, and then we opted for an unsupervised clustering approach to be applied in each group. Thus, we used the k-means algorithm to generate clusters.

In the next step, we applied the BERT transformers algorithm for each cluster to compute similarity measures between user stories.

We developed a web-based tool for visualizing clustering, similarity features, and UML diagrams. We used Python and the Flask framework to implement the tool. Fig. 1 shows the steps of our proposed method.

A. Grouping of user Stories by Actor

A user story is a very high-level definition of a requirement, containing just enough information; it is the most effective way to describe a requirement [34]. Agile project teams use one of these methods: Scrum, XP, Kanban, etc.

A user story often uses the following type of format:

As [actor], I want/I am able/I can [some objective], so that [some reason] [34].

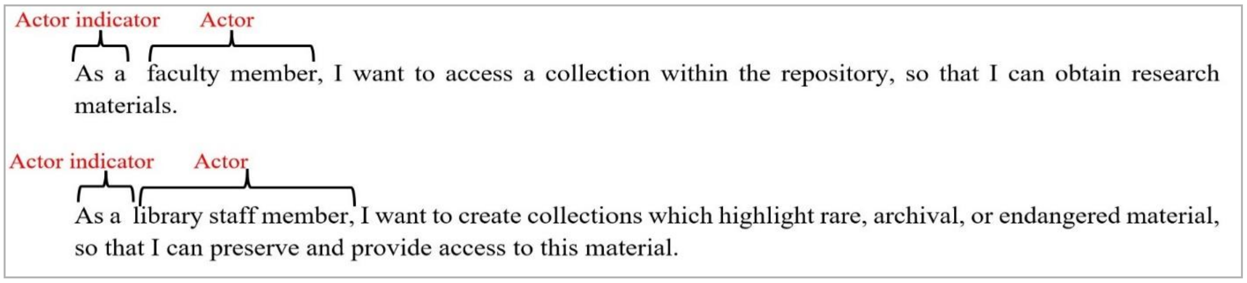
Before grouping the user stories, it is crucial to group the requirements by an actor. To achieve this goal, we extracted the actors using heuristic rules. The defined rules are based on tokenization and POS. We used the Python language and Spacy as NLP tool.

Since the extraction of a composite role composed of three words was not possible with the typed dependency in our previous work [10] [36], we switched to another method which consists of extracting the words following "As a/an": The role indicator is either an adjective or a noun, except pronouns. This solution allowed us to extract actor names independently of their name type. Our tool then used our defined rules to extract actors from user stories, which were stored in a text file. After that, it grouped the requirements by actor and saved each group separately. Fig. 2 shows examples of user stories.

Given these user stories:

- As an administrator, I want to search for a specific student, so that I can find the student records.
- As a student, I want to look up my student records, so that I can get a look at the details.

The two user stories are similar in the sense that they both aim to provide access to student records. They differ mainly in the actor involved: the administrator in the first case and the student in the second.



As a faculty member, I want to access a collection within the repository, so that I can obtain research materials.

As a library staff member, I want to create collections which highlight rare, archival, or endangered material, so that I can preserve and provide access to this material.

Fig. 1. Steps of our proposed approach.

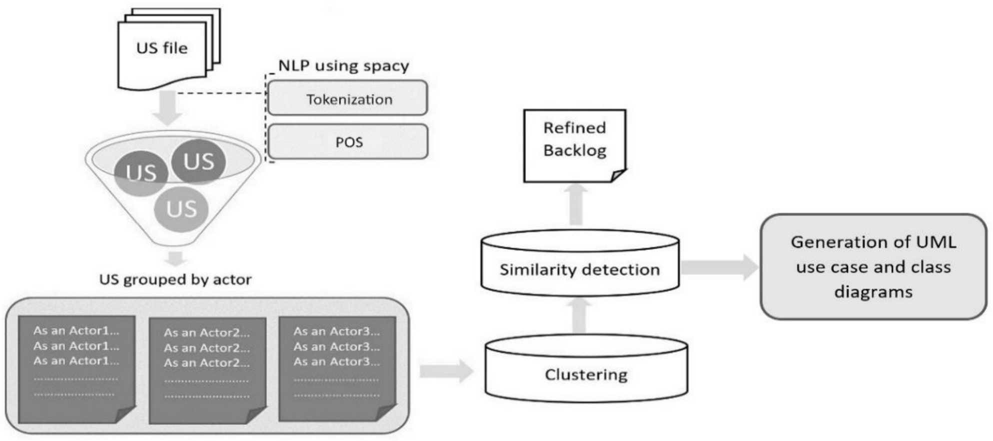


Fig. 2. Examples of user stories.

Unlike the methodology described in study [25], which focused on the identification of similarities between user stories without taking into account the “action executor” - i.e., the actor involved in the process - our approach took into account the specific roles or actors associated with the actions described in the stories.

Our method consisted of classifying user stories according to the actors involved, which makes it easier to keep similar stories describing identical actions carried out by different roles. These role-oriented requirements are the basis of the generated UML diagrams. For example, in a UML use case diagram, the two user stories could be represented as two separate but related use cases because they have similar functionalities (search and access to student records). They could be represented with different actors (the administrator and the student) but linked to a common use case, such as “View student records”. This would show the relationship between the two actions despite the differences between the actors.

B. *Clustering*

To implement our method, we used Sentence Transformers to embed sentences. Then, we employed the K-means algorithm to cluster the user stories based on their significance. We labeled the generated clusters using the Gensim library, which also provided us with keywords that describe the domain knowledge embedded in each cluster.

The Gensim library has an implementation of TF-IDF as part of its models, specifically the TF-IDF model module. This module converts documents into a matrix of token counts and calculates TF-IDF weights for each word.

These weights can then be used as features for tasks such as text classification, clustering, or similarity calculation.

To refine the backlog, we needed to eliminate duplicate user stories and display only relevant stories with their level of similarity. K-means clustering was used to achieve this by initially setting the value of k and examining the graph to identify the cut-off point where the slope changes. However, we opted for the silhouette calculation method to automate the k-value calculation at the outset. This approach enabled us to find the optimal k-value right from the start, making it easier to group user stories more efficiently. Then, we created a Python code to accomplish this task.

C. *User story Similarities and Opposite Meanings Detection*

1) Identification of similar user story pairs: To refine the backlog and detect duplicate user stories, we categorized the user stories by actor and then measured the similarity rate between each pair of user stories within each cluster. To achieve this goal, we used a sentence transformer (SBERT) to measure semantic similarity between the pairs of user stories. Below are the steps conducted for each cluster:

- Convert a sentence into a vector using sentence transformers.
- Convert several other sentences to vectors.
- Identify sentences with the smallest distance (Euclidean) or angle (cosine similarity) between them.
- Highlight the most similar user stories with rates above 75%.

We used SBERT to transform sentences into vectors, embedding them in a high-dimensional semantic space. Cosine similarity is then used to measure the degree of similarity between these vectors. More precisely, it assesses the comparability of documents regardless of their size by calculating the dot product of the vectors divided by the product of their Euclidean norms, as illustrated in Eq. (1).

$$\text{Similarity} = \cos(\theta) = \frac{A.B}{||A||||B||} = \frac{\sum_{i=1}^{n} \mathrm{A}_i \mathrm{B}_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \quad (1)$$

Here, A and B are two vectors. A.B is the dot product of those two vectors.

We aimed to select the most efficient sentence transformer that provides a higher percentage of similarity among pairs of sentences exhibiting similarities. We focused on comparing user stories that are similar but differ in their operations, ensuring that any dissimilarities were captured.

We adopted a structured two-phase approach to choose the most suitable model for measuring the similarity between pairs of user stories. First, we carried out a comparative analysis of

SBERT models to identify the most efficient model. Then, we extended our evaluation by comparing the SBERT model with other models available in the field.

On the one hand, Table I highlights the application of various SBERT models to detect similarities among pairs of user stories. This table presents the respective similarity rates achieved by applying different SBERT models to some pairs of user stories. On the other hand, Table III illustrates the comparison of the similarity rates obtained from different models, such as Elmo, Word2Vec, and a SBERT model, among others. Furthermore, Table II highlights the application of various SBERT models to detect similarities among pairs of user stories. This table presents the respective similarity rates achieved by applying different SBERT models to some pairs of user stories. Table I and Table II illustrate the two-level processes that form an integral part of our analysis approach.

Consider these user stories:

- US1: As a buyer, I am able to remove a product from the list.
- US2: As a buyer, I can drop a product from the list.
- US3: As a buyer, I can edit a product in the list.

Table II shows the similarity score of previous user stories by applying the sentence Transformers.

Based on the hierarchical structure of WordNet, Wup similarity evaluates the similarity between words according to their position in the lexical tree. Wup similarity does not take into account the contextual meaning of the words being compared unlike BERT, ELMO, and InferSent which integrate context to evaluate similarity. This lack of contextual consideration can limit Wup similarity's ability to capture semantic nuances in a variety of contexts.

TABLE I. SIMILARITY SCORES OF USER STORIES US1, US2, AND US3 USING SENTENCE TRANSFORMERS

Sentence Transformers	Similarity score US1-US2	Similarity score US1-US3
all-MiniLM-L6-v2	0.8180	0.6819656
all-MiniLM-L12-v2	0.8838	0.7911284
all-mpnet-base-v2	0.9025	0.8001925
all-distilroberta-v1	0.9006	0.7651011
paraphrase-mpnet-base-v2	0.9046	0.69480187

TABLE II. SIMILARITY SCORES OF USER STORIES US1, US2, AND US3 USING SIMILARITY MODELS

Models and similarity measures	Similarity score US1-US2	Similarity score US1-US3
USE	0.83511186	0.7759
SBERT(paraphrase-mpnet-base-v2)	0.9046	0.69480187
WORD2VEC	0.84041464	0.7834515
InferSent	0.9690	0.9582
ELMO	0.9342766	0.8595803
Wup similarity	0.66107734	0.3823969

Both Tables I and II illustrate that the sentence transformer “paraphrase-mpnet-base-v2” exhibits a considerable degree of similarity. We observe operations of US1 and US3, which involve different actions such as “remove” and “edit”. Therefore, the model chosen to evaluate similarity must be able to discern sentences with closely related but divergent meanings. This discernment is essential to avoid any negative impact on our analytical processes and backlog refinement.

To ensure that we had selected the appropriate Sentence Transformer model for our approach, we carried out an evaluation by testing several models on two distinct datasets. This evaluation aimed at determining the performance and effectiveness of each model in capturing the semantic meaning of user stories and generating high-quality sentence embeddings. To measure the performance of the models, we used evaluation metrics such as cosine similarity, accuracy, and F1-score. Through this evaluation process, we identified the model that consistently demonstrated superior performance and provided the most meaningful sentence embeddings.

We will provide further details of the evaluation in the next section.

2) Identification of the user story pairs with opposite meanings: When the similarity threshold is increased, the number of similar user story pairs decreases. To identify similar user stories, a minimum similarity rate of 75% is required while a much higher threshold with similarity rates exceeding 90% is necessary to detect duplicate user stories. In the first range, similar user stories were sometimes found where one story was included in another, providing additional information to the third part of the story. This was mainly due to the poor quality of the user stories. It is only by improving the quality of these stories that the occurrence of such similarities can be reduced.

However, in the second range, although we detected duplicate user stories, they sometimes had opposite meanings that were not captured by sentence similarity models.

Given these user stories:

- US4: As a buyer, I can add a product to the list.
- US5: As a buyer, I can not add a product to list.
- US6: As a user, I want backend changes for managing enum lists.
- US7: As a user, I want frontend changes for managing enum lists.

Table III displays the similarity scores of user stories with opposite meanings, which are US4 with US5, and US6 and US7.

In studying the ability of the SBERT model to detect pairs of opposing user stories, it was noted that when two user stories were presenting the same operation, with the same action verb in the format "As [actor], I want [action_verb]" and preceded by a negation "not", the model was able to identify their opposition. However, if they are not having the same operation, the SBERT model does not detect the dissimilarity.

Another case demonstrating opposite meanings is illustrated in the cases described in Table IV. Using sentence transformers, we observed a similarity score of 0.97 between user stories US6 and US7, highlighting substantial similarity despite contrasts in their content. To address this limitation, we developed specific rules. We used Typed dependencies provided by an NLP tool and the Wordnet API to search for synonyms of operations and checked for the presence of negation. Where negation was not explicitly expressed, we opted to detect antonyms for operations which improved the accuracy of detecting pairs of opposing user stories.

TABLE III. SIMILARITY SCORES OF USER STORIES WITH OPPOSITE MEANINGS

Models and similarity measures	Similarity score US4-US5	Similarity score US6-US7
USE	0.8806	0.9573
SBERT	0.42601973	0.9724
WORD2VEC	0.9803098	0.94255805
InferSent	0.9965	0.9912
ELMO	0.95751846	0.9578713
Wup similarity	0.6326058	0.6258217

As shown in Table III, we found a similarity of 0.97 between user stories US6 and US7 when using sentence transformers, suggesting high similarity within their differences. The WordNet approach was able to detect opposing meanings in some pairs of user stories. However, this method showed limited effectiveness for some examined pairs. The rules, we have developed, combine the use of the Wordnet API with typed dependency analysis to identify cases where user stories are initially identified as duplicates, having opposite meanings. These rules cover four different structures of user stories listed in Table IV.

The defined rules are based on the Stanford CoreNLP dependencies, such as neg, dobj, nmod, and advcl. This method includes the steps described in the proposed algorithm as follows.

Algorithm: Rules for negation extraction

```
1. Procedure(story1,story2):
2. Extract_part2(story1,story2)
3. Syn=Extract_synonyms(story1.verb1)
4. if [verb2 in Syn and check_negation(story1,story2)]
5.                     or are_antonyms(verb1, verb2) then
6. if dobj(verb1,obj1) = dobj(verb2,obj2)
7. or nmod(obj1,modifier)= nmod(obj2,modifier) then
8.                     return "negation found"
9. if story1.Action_advcl=story2.Action_advcl then
10.    return "negation found"
```

The algorithm described above is designed to detect negation in a pair of user stories. It worked by first extracting the second part of the user stories that presented the action. Then, it searched for synonyms of the first story's verb (verb1) and checked whether the second story's verb (verb2) is a synonym of the first story's verb (lines 1-3). In line 4, the check_negation(story1, story2) function was implemented by recognizing the negation indicator. In line 5, the algorithm examined whether the verb in story 1 is either a synonym preceded by a negation indicator or a direct antonym of the verb in story 2. If these conditions, along with the direct object or nominal modifier (nmod) had matched between the objects (lines 6-8), the algorithm returns "negation found". Line 9 checked whether the action in the "advcl" dependency is the same in both stories. If this condition, and the previous conditions, are met, the algorithm obtains the result "negation found". This condition is specific to Structure 4.

D. Similarity Detection in use Case Diagram

We used the Sentence Transformer technique to identify similarities between use cases in use case diagrams. Our approach involved concatenating the actor name and use cases, and then checking for similarities with a similarity score of over 90%, which was further improved by applying Wordnet for lemma synset interactions. This approach helped us extract similar use cases and user stories, which was very useful in creating refined models with minimal redundancy and inconsistency.

TABLE IV. STRUCTURES OF USER STORIES

Structures	User stories	Description	Example
Structure 1	As an actor, I want/I am able/I can **verb dobj**	An actor or role executes an action, which is described by a verb and involves a direct object.	As a customer, I want to view my order history so that I can track my purchases.
Structure 2	As an actor, I want/I am able/I can **verb dobj preposition nmod**	This structure includes additional detail about the direct object by adding a noun modifier.	As an administrator, I want to delete inactive user accounts more than 6 months old so that the database remains optimized.
Structure 3	As an actor, I want/I am able/I can **…** **preposition1 nmod1**… **preposition2**… **nmod2**	This structure adds a second nominal modifier to further clarify the objective of the user story.	As a manager, I want to include videos in courses for students.
Structure 4	As an actor, I want/I am able/I can **verb1 dobj1**… **preposition1 nmod preposition2 advcl**… **verb2 dobj2**…	It involves an initial action with a direct object, followed by a preposition and a modifier, leading to a second action with its direct object	As a manager, I want to generate reports of sales data by preserving product prices.

E. UML Diagrams Generation

After dividing the system into requirements by actors, we automated the process of generating UML models (class diagrams and use case diagrams) from the resulting clusters. To achieve this, we used various types of dependency present in NLP to implement specific NLP rules in the Prolog language to extract the elements making up the diagrams. Our first step involved generating the class diagram, followed by the use case diagram. Based on our previous work [11], we have developed Prolog rules to identify classes and associations. Facts were provided by an NLP tool that supplies nouns, verbs, and typed dependencies. The process of extracting class attributes followed the methodology described in study [10]. Hence, this method refined classes into attributes and converted certain associations into operations on classes. Prolog rules are presented as follows: Association(X, Y, Z); X is the name of the association, and Y and Z are the classes in the class diagram.

Extracting classes and associations is useful in generating use cases while association types, such as composition, helps determine the relationship between the use cases.

Given the importance of the terms used to represent the elements of the diagrams, refining the diagrams is essential as well. After the generation of UML diagrams, the next critical phase involved their refinement, which had been facilitated by the development of an ontology to store word equivalents [11].

F. Web-Based Framework

We developed a web-based tool that allows users to display similar user stories and their similarity percentages in selected clusters. Based on this percentage, users can decide whether to delete similar stories or to keep them. On the homepage, users can select a file containing user stories and click on a button to group them by actor. The file is automatically split into multiple ones, and users can then choose an actor from a drop-down list. Clustering by meaning is then performed, and the similarity rate is displayed. The results are presented in a table format, showing the similarity between each pair of user stories. We set the maximum similarity threshold at 75%. Once users have refined their backlog, they can generate UML class and use case diagrams to further improve the system design and development process.

V. Results and Discussion

This section presents the case studies and performance to evaluate the methodology of our approach for clustering, detecting more similar user stories, and generating a UML diagram for each cluster.

A. Evaluation

In this section, we present the evaluation of our approach. We firstly compare evaluating the "Paraphrase-mpnet-base-v2" model and the "all-MiniLM-L6-v2" model, which both belong to the SBERT models. We then compare our approach with other similarity detection models such as the Universal Sentence Encoder (USE), ELMO, and Word2vec. These models were evaluated for their performance in detecting similarities and distinguishing nuances in user stories. Therefore, we evaluated the UML class and use case diagrams that have been generated.

1) User stories clustering: In our study, we acknowledge the absence of a baseline or established benchmarks in the literature that specifically address the clustering of user stories using the k-means algorithm based on the "paraphrase-mpnet-base-v2" model.

It was therefore difficult to make a direct comparison with existing results or measurements. Although the lack of references limited our ability to quantitatively assess the performance of our approach, we solved this problem by carrying out a qualitative evaluation of the content of the groupings. To validate the effectiveness of the clusters in capturing similarities and relationships between user stories, a thoughtful manual evaluation was carried out. The latter was conducted by a team of experts who examined the logical consistency and relevance of the clusters. The findings of this evaluation validated the cluster quality and their ability to accurately represent the underlying patterns in the user stories.

2) Sentence transformer and word embedding models for similarity detection: The SBERT "Paraphrase-mpnet-base-v2" sentence transformer model was selected to measure similarities between user stories. This choice was based on its enhanced robustness in capturing the deep semantics of sentences, thus offering high-quality representations. We chose this model thanks to its ability to handle sentences with similar but different meanings, a crucial feature for in-depth analysis of user stories.

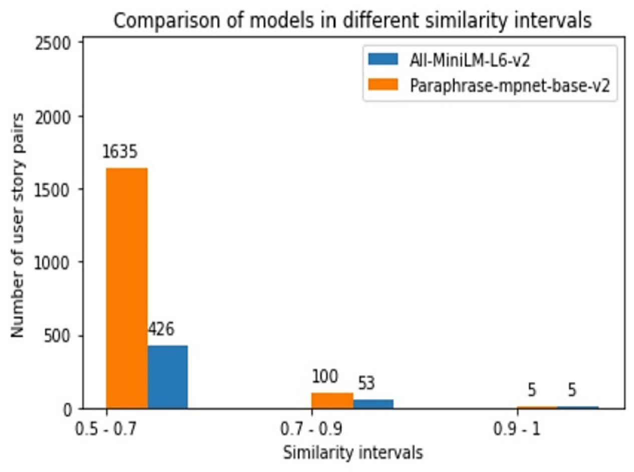
Course management and Archivespace, which are two distinct datasets, were used to evaluate Sentence transformer models. The first dataset, retrieved from the website Mountain Goat Software, consists of 102 user stories. The second dataset, from Archivespace, included 160 user stories. We collected these user stories as a part of the ArchiveSpace software development project.

We carried out evaluations to compare and assess the performance of the "all-MiniLM-L6-v2" and "Paraphrase-mpnet-base-v2" sentence transformer models, as well as the USE, ELMO, and Word2vec models, using several pairs of user stories. Table V presents the performance scores for the Archivespace dataset, including those of the sentence transformer and word embedding models.

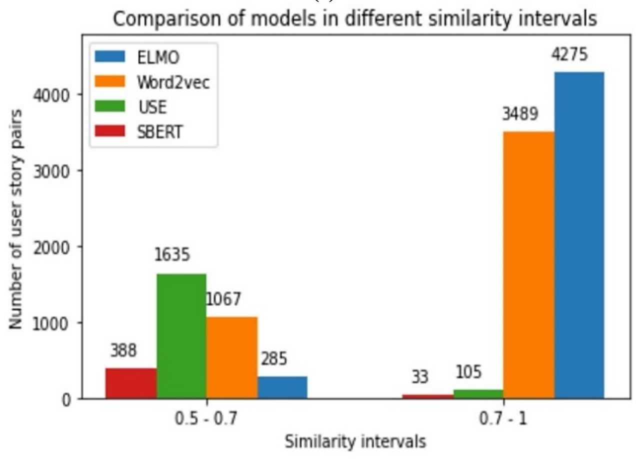
Fig. 3 illustrates the similarity ranges between pairs of user stories by using Paraphrase-mpnet-base-v2, All-MiniLM-L6-v2, USE, Word2vec, and ELMO.

TABLE V. THE PERFORMANCE SCORES OF SIMILARITIES BETWEEN USER STORIES USING SENTENCE TRANSFORMERS AND WORD EMBEDDING MODELS

Sentence transformer Model	**Score<0.5**	**0.5< score<0.7**	**0.7< score< 0.9**	**Score> 0.9**
All-MiniLM-L6-v2	4076	426	53	5
Paraphrase-mpnet-base-v2	2820	1635	100	5
USE	4139	388	31	2
Word2vec	4	1067	3462	27
ELMO	0	285	4234	41



(a)



(b)

Fig. 3. Comparison of paraphrase-mpnet-base-v2 and All-MiniLM-L6-v2 (a), and comparison of SBERT (Paraphrase-mpnet-base-v2), USE, Word2vec, ELMO (b) in different ranges.

While evaluating the Sentence Transformers models, we found significant differences between the two datasets. The “Paraphrase-mpnet-base-v2” and "All-MiniLM-L6-v2" models were initially tested for interval similarities greater than 0.9, and we discovered that these intervals gave almost identical results. However, when similarities were between 0.7 and 0.9, the evaluation of user story pairs presented challenges in determining whether they were similar or duplicates. In some cases, user stories appeared to be similar but turned out to be opposite or dissimilar. The results indicated that both models exhibited similar levels of similarity for most pairs of user stories. However, when considering slightly similar user stories, “Paraphrase-mpnet-base-v2” showed a distinguishing characteristic compared to the other model. It demonstrated a better ability to capture nuanced similarities and subtle differences between such user stories.

To compare the sentence transformer models with the use of USE, Word2vec, and ELMO, we conducted a second evaluation. The results revealed that USE detected lower similarity values than those detected by the sentence transformer models, even when the similarity interval was greater than 0.9. Furthermore, for pairs of user stories that should have a similarity rate below 0.8, USE often detected values above 0.8, particularly for user stories that appeared to be opposite or less similar. This divergence can be explained by saying that although user stories are similar in their action, differences in their description prevent them from being considered duplicated. Furthermore, the Sentence Transformers models, in particular the “paraphrase-mpnet-base-v2” model, showed better performance for the datasets used in this study. USE, while performing less well than SBERT, is still more reliable than Word2Vec and ELMO in assessing the similarity between these stories.

Further analysis revealed that Word2Vec, a widely used sentence embedding model, had limitations in detecting dissimilarity between user stories. Word2Vec may face challenges when distinguishing between user stories that are textually similar but involve different operations.

For example, it may struggle to discern the nuances between sentences such as “Users can add items to their shopping cart” and “Users can remove products from their shopping cart”, which is crucial in the context of software development.

In contrast, models such as Sentence Transformers are specifically designed to capture these contextual and operational nuances. They are highly accurate in measuring similarity while being more sensitive to subtle differences between similar user stories. In areas such as software engineering, where a precise understanding of requirements is essential, Sentence Transformers prove to be more effective for analyzing and planning software development.

After observing the similarity scores of different models in distinct ranges, notably between 0.5 and 0.7, as well as those above 0.7, we found that the SBERT model, in particular the "paraphrase-mpnet-base-v2" model, presented more consistent scores when compared to a manual approach. However, to better assess model choice, we used various metrics, including precision, recall, and F1 score [35]. Similarity identification was measured by accuracy, while coverage was measured by recall. We calculated precision, recall, and F1 score based on true positives, false positives, and false negatives. True positives (TP) corresponded to similarities correctly identified by our automated approach, while false positives (FP) referred to similarities incorrectly identified. A false negative (FN) was a labeled similarity not identified by an automated approach.

The evaluation metrics are measured as follows, as illustrated in Eq. (2), Eq. (3), and Eq. (4):

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2}$$

$$\text{Recall} = \frac{TP}{TP+FN} \tag{3}$$

$$\text{F1 Score} = \frac{2*\text{Precision}*\text{Recall}}{Precision+Recall} \tag{4}$$

Table VI shows the precision, recall, and F1 scores for each user story dataset for similarity pairs obtained using SBERT.

Table VII displays the similarity detection metric related to USE, Word2vec, and SBERT Models for User Story Pairs of the Archivespace dataset.

TABLE VI. EVALUATION METRICS FOR SIMILARITY DETECTION OF USER STORIES OF THE FIRST DATASET

Models	Similarity of user stories pairs					
	TP	FP	FN	Precision	Recall	F1-score
ELMO	7	13	3	35%	70%	46%
Word2vec	14	65	0	17%	100%	29%
USE	5	1	6	83%	45%	58%
SBERT	14	1	0	93%	100%	96%

TABLE VII. COMPARISON OF SIMILARITY DETECTION MODELS FOR USER STORY PAIRS OF ARCHIVE SPACE DATASET

Models	Similarity of user stories pairs					
	TP	FP	FN	Precision	Recall	F1-score
ELMO	7	13	3	35%	70%	46%
Word2vec	14	65	0	17%	100%	29%
USE	5	1	6	83%	45%	58%
SBERT	14	1	0	93%	100%	96%

Table VI and Table VII show similar user stories with similarity rates above 90%. When the similarity rate exceeds 90%, the user stories are more similar. The low similarity metrics obtained using Word2Vec to evaluate user stories can be explained by the fact that this model does not capture the semantic complexity of texts as effectively as more recent approaches such as USE and SBERT.

In some cases, one user story was included in another one which added more information in the third part of the user story which was due to poor writing of user stories. If the user stories were of high quality, they could reduce the occurrence of similar user stories. In other cases, the approach detected two similar user stories with opposite meanings. However, by using Wordnet and our defined rules, we have overcome this shortcoming.

Grouping user stories by actor before detecting similarity between pairs of stories had been found to help address the problem of false positives that can occur when multiple actors perform the same action. If stories were not grouped by actor, the similarity approach may identify false similarities between two stories that had different actors. However, the Universal Sentence Encoder (USE) model is limited in detecting explicit negation. Because of relying on language patterns, it can struggle to capture the opposite or contradictory meaning introduced by negation words like “not”. This can result in inaccurate detection of differences in meaning when negation is present.

3) Evaluation of UML class and use case models: During the evaluation process, our primary focus was to compare the UML use case diagrams generated by the proposed approach with the manual UML use case diagrams created by our team of experts. These diagrams were based on user stories. We aimed to assess the accuracy and effectiveness of the generated models in comparison to the manual approach and a relevant existing approach in the field. However, we were unable to make a direct comparison between our results and those of the referenced article because the identical case study used by the authors was unavailable.

To assess the quality and relevance of the UML use case models, we implemented a manual evaluation approach. Our team of experts carefully examined the models, comparing them with the corresponding user stories to ensure accuracy and consistency. In addition, to evaluate class diagrams, we used the same case study as [4], enabling a comparative analysis of the approach to extracting artifacts from class diagrams. This evaluation involved examining the artifacts extracted and comparing them with the expected artifacts derived from the case studies. We considered aspects such as completeness, correctness, and relevance to assess the effectiveness of our approach.

- Case study

The user stories, in this case study, represented event management: booking and purchasing an event ticket [36].

Tokenization, lemmatization, stemming, and part-of-speech (POS) analysis were used to process user stories. Typed dependencies were then applied to each user story. The extraction of design components was based on defined rules, which relied on the dependencies that were exploited and analyzed.

The final results of the extraction of design elements for the given user story are presented in the tables below. Classes and their attributes are listed in Table VIII.

TABLE VIII. CLASSES AND THEIR ATTRIBUTES

Classes	Attributes
Account	Password
Visitor	Personal_details
Ticket	Price
Ticket	Type

The relationship results are shown in Table IX.

TABLE IX. RELATIONSHIPS RESULTS

Relationships
Create (account, Visitor)
Have (account, Visitor)
Rename (account, Visitor)
Choose (event, Visitor)
Search (event, Visitor)
See (event, Visitor)
Choose (payment_methods,Visitor)
Buy (ticket, Visitor)
Book (ticket, Visitor)
Purchase (ticket, Visitor)
Receive (ticket, Visitor)
Have (ticket, event)

Table X indicates the results of the operations.

TABLE X. GENERATION OF CLASS OPERATIONS

Classes	Operations
Visitor	Provide (personal_details)
Account	Change(password)
Event	Filter(type)
Ticket	See (price)
Ticket	Choose(type)
Event	Choose(type)

The Visual Narrator tool [4] identified several classes, including Visitor, Account, System, Event, Ticket, EventType, Type, AccountPassword, Password, TicketPrice, Price, Detail, and Method. However, there are some relationships that the tool extracted, but our approach did not, such as hasType (Event, EventType), hasPrice (Ticket, TicketPrice), and hasPassword (Account, AccountPassword). Additionally, the tool detected that Visitors and Systems can log in and log out, but our approach did not.

The Visual Narrator tool [4] uses an approach that creates numerous compound classes and inheritance relationships. However, this method can result in complex classes and inheritance relationships that are unnecessary due to the absence of attribute extraction rules. Table XI compares the total items detected by [4], our approach, and manual analysis.

TABLE XI. THE TOTAL OF DESIGN ELEMENTS DETECTED BY [4] AND OUR APPROACH [36]

	Actors	Classes/ Entity	Attributes	Relations-hips	Operations
[4]	1	13	0	19	0
Our approach	1	5	5	12	6
Manually	1	5	5	12	6

Our approach, in the case studies, demonstrated significant improved performance compared to the method described in [4], achieving a high precision rate of 98% when comparing these results to those obtained manually.

B. Results

We provided a detailed explanation with the help of figures to clarify the process of clustering and generating UML diagrams using our tool.

Fig. 4 shows a web page that allows the user to download a file containing a set of user stories. Once downloaded, we performed the clustering by actor by clicking the “Clustering by actor” button. The extracted actors were then included in a drop-down list. To perform clustering by meaning, the user needed to click on the “Analyze” button, as illustrated in Fig. 5. It is worth noting that a text field to specify the number of clusters was not included from the beginning. This process was automated based on silhouette calculations.

Fig. 5 displays the clusters in a table, along with keywords representing the cluster's meaning, as well as links to select similar user stories. This table allows the user to easily navigate through the clusters and select the most relevant user stories.

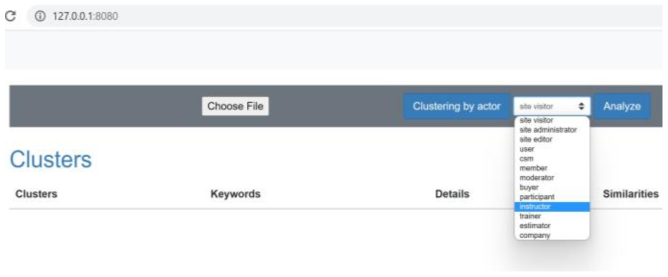
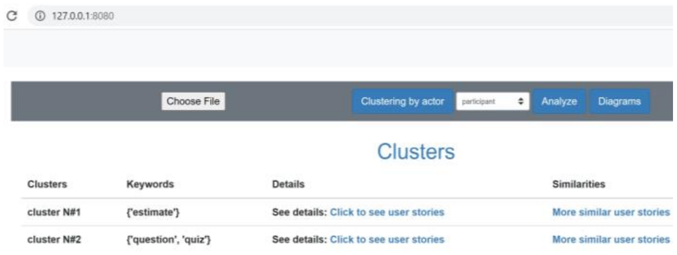


Fig. 4. Items generated in the drop-down list from the user stories file.

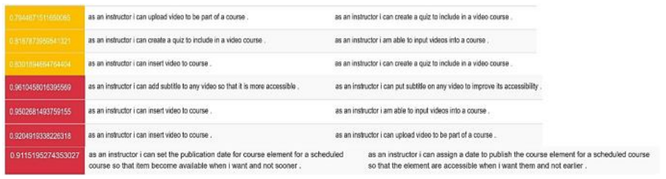


Clusters

Clusters	Keywords	Details	Similarities
cluster N#1	{'estimate'}	See details: Click to see user stories	More similar user stories
cluster N#2	{'question', 'quiz'}	See details: Click to see user stories	More similar user stories

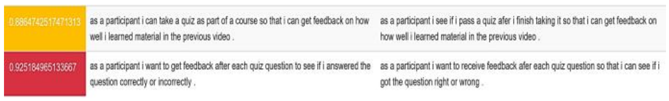
Fig. 5. Clusters generated corresponding to Participant actor.

Fig. 6 and 7 show the similarity scores generated for pairs of user stories within the cluster of the Instructor and Participant actors. By highlighting the most similar user stories, our tool facilitated the decision to delete or keep them.



0.7944871511650065	as an instructor i can upload video to be part of a course .	as an instructor i can create a quiz to include in a video course .
0.8187873959541321	as an instructor i can create a quiz to include in a video course .	as an instructor i am able to input videos into a course .
0.8301894684764404	as an instructor i can insert video to course .	as an instructor i can create a quiz to include in a video course .
0.9610458016395569	as an instructor i can add subtitle to any video so that it is more accessible .	as an instructor i can put subtitle on any video to improve its accessibility .
0.9502681493759155	as an instructor i can insert video to course .	as an instructor i am able to input videos into a course .
0.9204919338226318	as an instructor i can insert video to course .	as an instructor i can upload video to be part of a course .
0.9115195274353027	as an instructor i can set the publication date for course element for a scheduled course so that item become available when i want and not sooner .	as an instructor i can assign a date to publish the course element for a scheduled course so that the element are accessible when i want them and not earlier .

Fig. 6. Similarity scores generated for pairs of user stories within the cluster of the Instructor actor.



0.8864742517471313	as a participant i can take a quiz as part of a course so that i can get feedback on how well i learned material in the previous video .	as a participant i see if i pass a quiz afer i finish taking it so that i can get feedback on how well i learned material in the previous video .
0.925184965133667	as a participant i want to get feedback after each quiz question to see if i answered the question correctly or incorrectly .	as a participant i want to receive feedback afer each quiz question so that i can see if i got the question right or wrong .

Fig. 7. Similarity scores generated for pairs of user stories within the cluster of the Participant actor.

Finally, Fig. 8-12 display the generated class and use cases UML diagrams corresponding to the “Participant” and “Instructor” actors. These diagrams provide a clear and consistent understanding of the product requirements, making it more convenient for team members to collaborate and work efficiently.

As can be seen in Fig. 10 and 12, using sentence transformers allows for the detection of duplicate use cases such as “receive feedback” and “get feedback”, the same for the “receive” and “get” associations between the participant and feedback class. Regarding the use cases “upload PDFS” and "download PDFS" these components were similar in using sentence transformation yet using the Wordnet approach shows that they had opposite meanings. We believe that combining both approaches can achieve better performance.

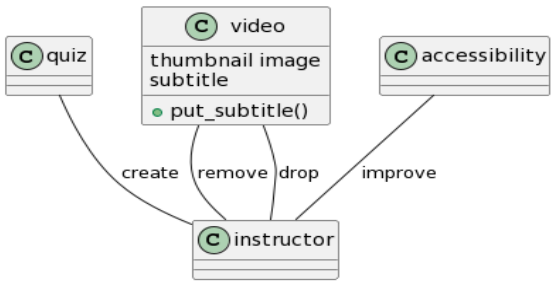


Fig. 8. Generated the UML class diagram corresponding to the instructor's cluster.

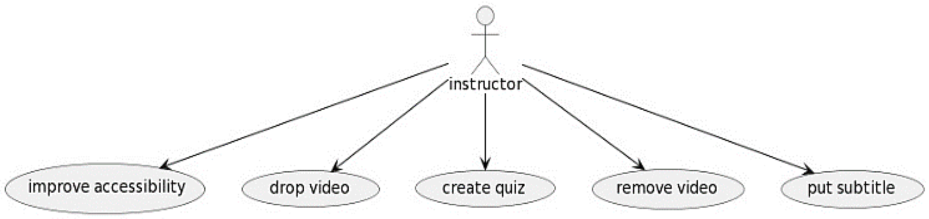
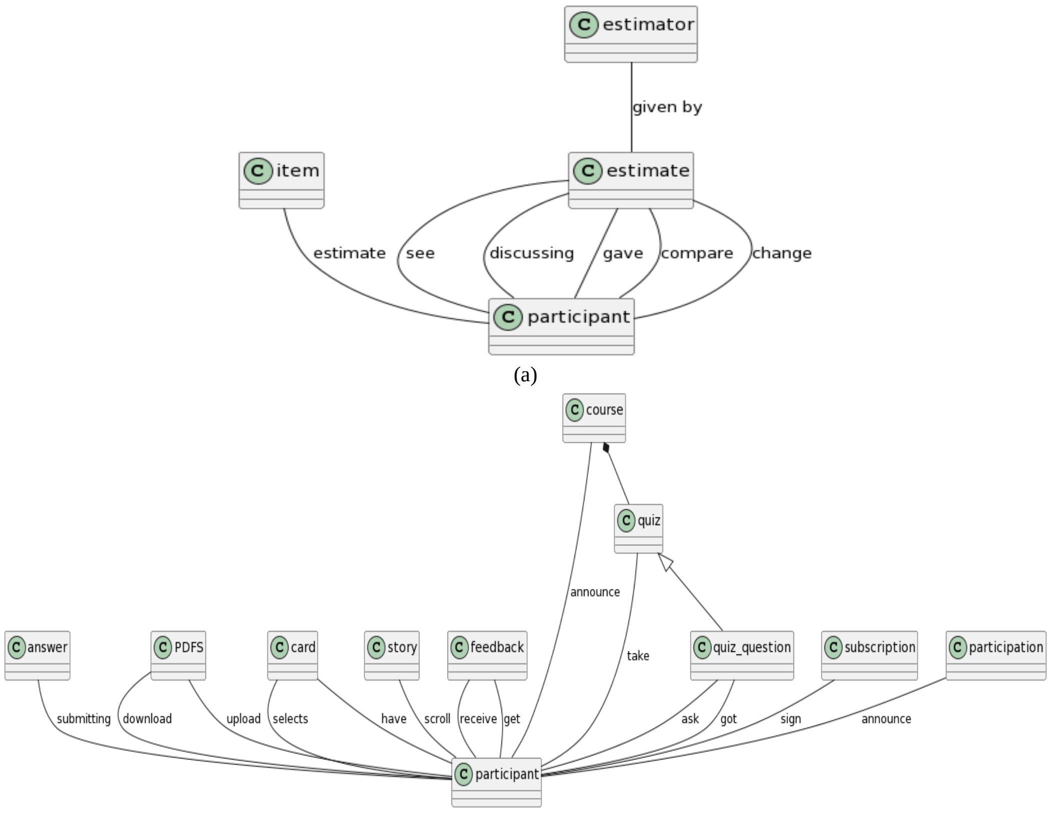


Fig. 9. Generated the UML use case diagram corresponding to the instructor's cluster.



(a)

(b)

Fig. 10. Generated the UML class diagrams corresponding to participant's clusters #1 (a) and #2 (b).

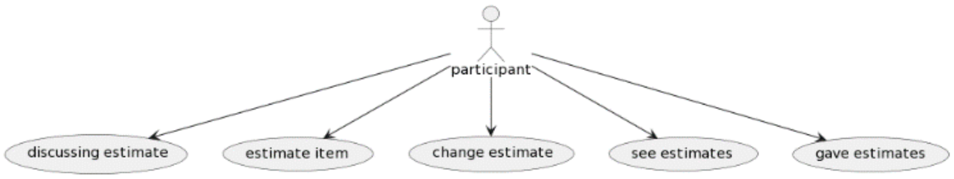


Fig. 11. Generated the UML use case diagram corresponding to participant's cluster #1.

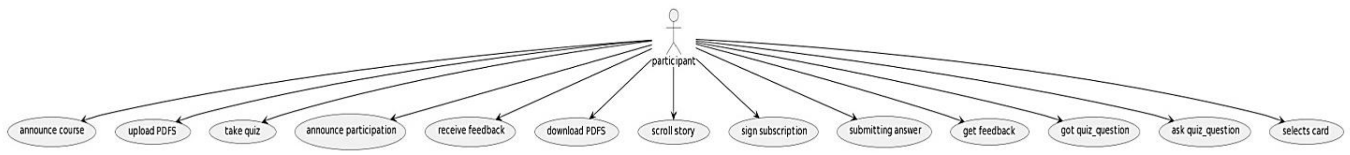


Fig. 12. Generated the UML use case diagram corresponding to Participant's cluster #2.

C. Threats to Validity

In this section, we will discuss the possible threats to the validity of our proposed approach.

- Internal Validity: The quality of user stories can have a significant impact on the results, particularly if they are poorly written or contain non-functional scenarios. This can negatively affect the relationships between classes and generate inadequate UML diagrams.

To overcome this problem, it is essential to ensure the quality and functional accuracy of user stories.

- Construct Validity: The rules defined for assessing similarities between user stories may be limited, as they only capture a restricted set of sentence structures in the context of opposite meanings. It is necessary to have a more comprehensive analysis that considers the semantic meaning and context of the user stories to accurately identify and handle potential contradictions.

- External Validity: In evaluating our approach, we analyzed two case studies to determine the effectiveness of our approach in detecting similar user stories. Although the initial results were promising, there is a need to extend our evaluation to a larger number of case studies.

D. Discussion

In this section, we compare our approach with existing state-of-the-art techniques [14], [16] that also use user stories as input. Although these approaches focus on clustering and duplicate detection, they do not include diagram generation. Another study [24] focused on generating use case diagrams from user stories, but it only considered user stories with a simple structure and did not address duplicate or similarity detection. In [25] Detection of similarities in user stories is ineffective because their method is unable to distinguish between operations such as "delete" and "add" in stories. These operations, which represent the core functionality, remain indistinguishable. The model used to detect similar user stories, namely USE, lacks the ability to discern negations in these stories, such as "not add" and "add".

In contrast, our approach built on our previous work [10]-[12],[36], in which we used NLP techniques to generate various UML diagrams from user stories. The paper [36] expands upon the content of [10].

The previous approach [11], based on ontology, Prolog rules, and WordNet synsets, focused on refining UML diagrams by defining explicit relationships and using domain-specific vocabulary. It addressed redundancy and duplicate detection to some extent, but it had limitations. Maintaining and updating the ontology with relevant vocabulary posed challenges. Focusing on explicit definitions might overlook subtle nuances in redundancy and duplication. Additionally, the approach required extensive domain knowledge and manual refinement.

In contrast, the new approach incorporated AI techniques, in particular SBERT models and clustering to refine the generated backlog and UML diagrams. It used machine learning to capture semantic similarities as well as the rules we defined to achieve better results. This enabled duplicates to be detected without the need to explicitly define an ontology.

Our approach focused on improving the refinement process to enhance the quality and accuracy of the generated models. The backlog was refined using clustering and similarity detection techniques before generating the UML diagrams. This step helped in handling the large number of user stories present and guaranteed the accuracy of the generated diagrams.

Refining the use case diagrams and detecting similar use cases made our approach more complete and refined compared to other approaches used by different authors.

Table XII summarizes the relevant related works.

Table XIII presents a comparison between the previous approach and our proposed approach.

In comparison to old approaches, our approach offered several significant advantages using automatic refinement of UML diagrams. Firstly, by integrating prior refinement of the user story backlog, early detection and elimination of redundancies in the process can be achieved. This allowed us to create more concise, better organized, and more relevant UML diagrams to represent the system's functionalities.

Secondly, through using AI techniques, particularly SBERT models, our approach offered better detection of duplications and similarities between user stories. Clustering user stories and subsequently labeling these clusters, allowed for an efficient backlog structuring and an improved organization. The refinement process eliminates redundant information and similar functionalities with great precision, resulting in clearer and more readable UML diagrams.

Automating the backlog refinement process saves the team valuable time. Employing the AI-based prototype allows for quick and accurate execution of tasks such as similarity detection, user story clustering, and diagram generation.

TABLE XII. SUMMARY OF RELEVANT LITERATURE

Approach	Models and tools	Input	Output
[14]	K-means clustering algorithm applied to user stories.	user stories	Clusters of user stories
[16]	- Semantic similarity measures to suggest possible cases of duplication between user stories. - Analysis of semantic similarity measures based on the WordNet lexical database, in particular WuP similarity.	user stories	Determine the level of similarity among user stories
[24]	- Agglomerative Hierarchical Clustering (AHC) algorithm to group requirements into clusters. - Use of the Gensim API to extract keywords by group. - Definition of simple NLP rules for component extraction to generate a use case diagram	user stories	Use case diagram
[25]	- The model USE for calculating similarity - For app development: Laravel and React. - Manual approach to detect similarities greater than 60%	User stories	- Estimate efforts and costs for agile projects: Time spent on similar past projects - Similarity user stories detection
Our approach	- Flask - Python - SBERT models - Defining NLP rules to identify every dissimilar previously classified as similar by SBERT models	User stories	- Similarity user stories detection - Clustering and labeling each cluster - UML diagram generation

TABLE XIII. COMPARISON BETWEEN THE PREVIOUS APPROACH AND OUR PROPOSED APPROACH

Features	Old Approach	New Approach
Refinement Method	Prolog rules, ontology, WordNet synsets	Clustering, SBERT models, and definition of rules
Refinement Stage	Post-generation refinement of UML diagrams	Initial backlog refinement to detect and eliminate redundancy
Contextual Meaning	Not considered	Contextual meaning detection with SBERT models
Backlog Refinement	Not addressed	Initial backlog refinement to detect similar user stories
Duplicate Detection	Limited capability in detecting duplicates and similarities	Improved accuracy in identifying duplicate user stories through advanced AI techniques

VI. CONCLUSION

In Agile project management, Backlog refinement is a crucial process. It aims to ensure that the backlog contains prioritized and well-defined user stories. However, refining the backlog using a traditional manual approach is time-consuming and prone to errors.

In this paper, we proposed an approach to refine the backlog by detecting similar user stories with a percentage that will help the designer decide to delete or leave the concerned user stories. Additionally, we aimed to reduce the occurrence of similar use cases in the use case diagram UML. Our proposed approach combined clustering and duplicate detection to automatically generate UML diagrams from a set of refined user stories in each cluster. To achieve this, we used the K-means algorithm to cluster similar user stories. In addition, we incorporated the SBERT model to measure the similarity between these user stories and use cases. Using multiple pairs of user stories, the case studies conducted show that our proposal achieves high performance.

In future work, we plan to further improve our approach by using multiple datasets to improve performance. Furthermore, we aim to define more rules to detect opposite meanings in user stories. Finally, we will focus on detecting non-functional requirements and generating acceptance criteria from them to improve the quality of user stories. It is essential to ensure that user stories remain well-defined and focused on functional aspects, while keeping non-functional requirements, such as performance, security, and usability constraints, specified in the acceptance criteria. By addressing these challenges, we can further enhance the accuracy and efficiency of requirements engineering in software development, ultimately leading to an overall improvement in product quality.

REFERENCES

[1] Belani H, Vukovic M, Car Z. Requirements engineering challenges in building AI-based complex systems. Proceedings - 2019 IEEE 27th International Requirements Engineering Conference Workshops, REW 2019, Institute of Electrical and Electronics Engineers Inc.; 2019, p. 252–5.

[2] Yang C, Liang P, Avgeriou P. A systematic mapping study on the combination of software architecture and agile development. Journal of Systems and Software 2016;111:157–84. https://doi.org/10.1016/j.jss.2015.09.028.

[3] Chantit S, Essebaa I. Towards an automatic model-based scrum methodology. Procedia Comput Sci, vol. 184, Elsevier B.V.; 2021, p. 797–802. https://doi.org/10.1016/j.procs.2021.03.099.

[4] Lucassen G, Robeer M, Dalpiaz F, van der Werf JMEM, Brinkkemper S. Extracting conceptual models from user stories with Visual Narrator. Requir Eng 2017;22:339–58. https://doi.org/10.1007/s00766-017-0270-1.

[5] Javed M, Lin Y. Iterative process for generating ER diagram from unrestricted requirements. ENASE 2018 - Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering2018;2018-March:192–204. https://doi.org/10.5220/0006778701920204.

[6] Y. Rhazali, Y. Hadi, I. Chana, M. Lahmer, and A. Rhattoy, “A model transformation in model driven architecture from business model to web model.” IAENG International Journal of Computer Science, vol. 45, no.

1, pp. 104–117, 2018. [Online]. Available: https://www.researchgate.net/publication/323275958.

[7] Jaiwai M, Sammapun U. Extracting UML Class Diagrams from Software Requirements in Thai using NLP. 2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE). IEEE, 2017. p. 1-5. https://doi.org/10.1109/JCSSE.2017.8025938.

[8] Abdelnabi EA, Maatuk AM, Hagal M. Generating UML Class Diagram from Natural Language Requirements: A Survey of Approaches and Techniques. 2021 IEEE 1st International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering, MI-STA 2021 - Proceedings, Institute of Electrical and Electronics Engineers Inc.; 2021, p. 288–93. https://doi.org/10.1109/MISTA52233.2021.9464433.

[9] Yang S, Sahraoui H. Towards automatically extracting UML class diagrams from natural language specifications. Proceedings - ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems, MODELS 2022: Companion Proceedings, Association for Computing Machinery, Inc; 2022, p. 396–403. https://doi.org/10.1145/3550356.3561592.

[10] Nasiri S, Rhazali Y, Lahmer M, Chenfour N. Towards a Generation of Class Diagram from User Stories in Agile Methods. Procedia Comput Sci 2020;170:831–7. https://doi.org/10.1016/j.procs.2020.03.148.

[11] Nasiri S, Rhazali Y, Lahmer M, Adadi A. From User Stories to UML Diagrams Driven by Ontological and Production Model. International Journal of Advanced Computer Science and Applications, vol. 12, no. 6, 2021. https://doi.org/10.14569/IJACSA.2021.0120637.

[12] Nasiri S, Adadi A, Lahmer M. Automatic generation of business process models from user stories. International Journal of Electrical and Computer Engineering 2023;13:809–22. https://doi.org/10.11591/ijece.v13i1.pp809-822.

[13] Salman HE, Hammad M, Seriai AD, Al-Sbou A. Semantic clustering of functional Requirements using agglomerative hierarchical clustering. Information (Switzerland) 2018;9. https://doi.org/10.3390/info9090222.

[14] Kumar B, Tiwari UK, Dobhal DC, Negi HS. User Story Clustering using K-Means Algorithm in Agile Requirement Engineering. 2022 International Conference on Computational Intelligence and Sustainable Engineering Solutions (CISES), 2022, p. 1–5. https://doi.org/10.1109/CISES54857.2022.9844390.

[15] R. Selva Birunda S. and Kanniga Devi. Review on Word Embedding Techniques for Text Classification. Innovative Data Communication Technologies and Application, 2021, pp. 267–281.

[16] Barbosa R, Silva AEA, Moraes R. Use of Similarity Measure to Suggest the Existence of Duplicate User Stories in the Scrum Process. Proceedings - 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN-W 2016, Institute Of Electrical and Electronics Engineers Inc.; 2016, p. 2–5. https://doi.org/10.1109/DSN-W.2016.27.

[17] Wang J, Dong Y. Measurement of text similarity: a survey. Information. 2020 Aug 31;11(9):421.

[18] Banerjee S, Pedersen T. An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet. International Conference on Intelligent Text Processing and Computational Linguistics, pp. 136-145, 2002. https://doi.org/10.1007/3-540-45715-1_11.

[19] F. S. Bäumer MG. Running out of words: How similar user stories can help to elaborate individual natural language requirement descriptions. Commun. Comput. Inf. Sci., vol. 639, pp. 549-558, Oct. 2016. https://doi.org/10.1007/978-3-319-46254-7.

[20] Wautelet Y, Heng S, Kolp M, Mirbel I. Unifying and extending user story models. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 2014;8484 LNCS:211–25. https://doi.org/10.1007/978-3-319-07881-6_15.

[21] Rodeghero P, Jiang S, Armaly A, Mcmillan C. Detecting User Story Information in Developer-Client Conversations to Generate Extractive Summaries. Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017, pp. 49–59, https://doi.org/10.1109/ICSE.2017.13.

[22] Dalpiaz F, van der Schalk I, Lucassen G. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 10753 LNCS, Springer Verlag; 2018, p. 119–35. https://doi.org/10.1007/978-3-319-77243-1_8.

[23] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," Aug. 2019, [Online]. Available: http://arxiv.org/abs/1908.10084.

[24] T. Kochbati, S. Li, S. Gérard, and C. Mraidha, "From user stories to models: A machine learning empowered automation," in *MODELSWARD 2021 - Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development*, SciTePress, 2021, pp. 28–40. doi: 10.5220/0010197800280040.

[25] A. Grzegorz Duszkiewicz, J. Glumby Sørensen, N. Johansen, H. Edison, and T. Rocha Silva, "On Identifying Similar User Stories to Support Agile Estimation based on Historical Data," 2022. [Online]. Available: https://www.sdu.dk/staff/hedis.

[26] C. Wu, C. Wang, T. Li, and Y. Zhai, "A Node-Merging based Approach for Generating iStar Models from User Stories," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, Knowledge Systems Institute Graduate School, 2022, pp. 257–262. doi: 10.18293/SEKE2022-176.

[27] T. Wang, C. Wang, T. Li, Z. Liu, and Y. Zhai, "User Story Quality Assessment Based on Multi-dimensional Perspective: A Preliminary Framework," 2022. [Online]. Available: http://ceur-ws.org.

[28] R. Selva Birunda S. and Kanniga Devi, "A Review on Word Embedding Techniques for Text Classification," in Innovative Data Communication Technologies and Application, A. M. and B. R. and B. Z. A. Raj Jennifer S. and Iliyasu, Ed., Singapore: Springer Singapore, 2021, pp. 267–281.

[29] A. Jalilifard, V. F. Caridá, A. F. Mansano, R. S. Cristo, and F. P. C. da Fonseca, "Semantic Sensitive TF-IDF to Determine Word Relevance in Documents," Jan. 2020, doi: 10.1007/978-981-33-6977-1.

[30] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information", 2017, doi: 10.1162/tacl_a_00051/1567442/tacl_a_00051.pdf.

[31] E. M. Dharma, F. Lumban Gaol, H. Leslie, H. S. Warnars, and B. Soewito, "The Accuracy Comparison Among Word2vec, Glove, And Fasttext Towards Convolution Neural Network (Cnn) Text Classification," J Theor Appl Inf.

[32] D. Cer et al., "Universal Sentence Encoder," Mar. 2018, [Online]. Available: http://arxiv.org/abs/1803.11175.

[33] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data Clustering: A Review," 2000.

[34] M. Cohn, User Stories Applied: For Agile Software Development. 2004.

[35] H. Schütze, C. D. Manning and P. Raghavan, Introduction to Information Retrieval, Cambridge, U.K.:Cambridge University Press, 2008.

[36] S. Nasiri, Y. Rhazali, and M. Lahmer, "Towards a Generation of Class Diagram From User Stories in Agile Methods," 2021, pp. 135–159. doi: 10.4018/978-1-7998-3661-2.ch008.