From User Stories to Models: A Machine Learning Empowered Automation

Takwa Kochbati, Shuai Li, Sébastien Gérard and Chokri Mraidha
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Keywords: User Story, Machine Learning, Word Embedding, Clustering, Natural Language Processing, UML Use-case.

Abstract: In modern software development, manually deriving architecture models from software requirements expressed in natural language becomes a tedious and time-consuming task particularly for more complex systems. Moreover, the increase in size of the developed systems raises the need to decompose the software system into sub-systems at early stages since such decomposition aids to better design the system architecture. In this paper, we propose a machine learning based approach to automatically break-down the system into sub-systems and generate preliminary architecture models from natural language user stories in the Scrum process.

Our approach consists of three pillars. Firstly, we compute word level similarity of requirements using word2vec as a prediction model. Secondly, we extend it to the requirement level similarity computation, using a scoring formula. Thirdly, we employ the Hierarchical Agglomerative Clustering algorithm to group the semantically similar requirements and provide an early decomposition of the system. Finally, we implement a set of specific Natural Language Processing heuristics in order to extract relevant elements that are needed to build models from the identified clusters.

Ultimately, we illustrate our approach by the generation of sub-systems expressed as UML use-case models and demonstrate its applicability using three case studies.

1 INTRODUCTION

In order to improve their performance, organizations tend to shift to Agile methodologies (Lindvall et al., 2002). A key feature of their success is that they empower the customer-developer relationship in order to develop quicker, better and higher-quality products. In order to meet this goal, Agile methodologies such as Scrum (Schwaber, 1997) rely on incremental, iterative and adaptive processes. The first objective in the Scrum process is to define the product backlog, that is, a list of elicited and prioritized requirements that should be implemented. Then, Scrum processes in sprints. Each sprint contains the tasks to achieve by the developer teams.

Requirements analysis is essential for the success of this process as developers need to understand requirements before moving to the implementation phase (Verner et al., 2005). In this context, user stories are increasingly adopted to express requirements in Scrum (Kassab, 2015; Lucassen et al., 2016a). They represent a high-level definition of requirements expressed in semi-structured natural language notation. Many textual templates have been proposed for user stories. The one that is most used by practitioners has the form of: " *As a <type of user> I want <some goal> so that <some reason>*" (Cohn, 2004).

Since the size of the developed systems increases due to the increasing number of partners involved in the process, the number of user stories consequently grows. This growth in number raises the need to break-down the system into a set of sub-systems, each covering a set of semantically similar user stories. Such early decomposition helps the developers to better understand and realize the target software and it may be a basis to build the product backlog. Eventually, each sub-system needs to be expressed in a semi-formal manner such as visual models in order to keep stakeholders involved in the requirements engineering tasks (Alexander and Maiden, 2004).

In this context, the Unified Modeling Language (UML) (Booch et al., 2015) has been employed as a visual language that supports requirements engineering. It has been widely used to specify the features of software systems, and to reduce the ambiguity between the requirement specifications and the design,

Kochbati, T., Li, S., Gérard, S. and Mraidha, C.
From User Stories to Models: A Machine Learning Empowered Automation.
DOI: 10.5220/0010197800280040
In *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2021)*, pages 28-40
ISBN: 978-989-758-487-9

on the basis of models. Among these models, use-cases have been widely used to capture the requirements from the user's point of view. In addition, they are easy to understand and provide an excellent way for communicating (Bittner, 2002).

However, manually grouping similar user stories and generating UML models is a tedious and time-consuming task especially for large scale systems. Furthermore, the integration of model-based approaches in Scrum process has been hard to perform due to the lack of powerful automation tools as well as the focus of teams on the implementation rather than analysis or documentation (Löffler et al., 2010).

Several studies found in literature have focused on automating models generation from natural language software requirements (Elallaoui et al., 2015), (Elallaoui et al., 2018), (Arora et al., 2016). Other works focused on the clustering of natural language requirements with the goal of decomposing the target system at early stages (Barbosa et al., 2015), (Amannejad et al., 2014). On the one hand, we observed that current requirement clustering approaches are lacking accuracy and fail to achieve a high degree of automation. On the other hand, using requirement clustering as a first step to automatically derive models has never been considered in the literature.

In order to overcome these limitations, we propose a machine learning based approach to automatically break-down the target system into a set of subsystems as a first step towards architecture models generation. The core of our approach is a clustering solution that groups natural language user stories based on their semantic similarity. Accordingly, our work promotes the integration of model-based approaches in Scrum.

Specifically, in this paper we make the following contributions: 1) we employ a word embedding model, word2vec, as a prediction model to compute word level semantic similarity as it has shown to outperform traditional counting models (Baroni et al., 2014). Then, we extend it to the requirement level using a scoring formula for text similarity; 2) we use the Hierarchical Agglomerative Clustering algorithm (HAC) to cluster natural language user stories based on their semantic similarity. As the selection of the number of clusters is not straightforward and requires manual intervention, we implement an operation to automatically estimate the optimal number of clusters ; 3) finally, we implement a set of specific Natural Language Processing (NLP) heuristics in order to extract relevant elements that are needed to build the required UML use case models from the identified clusters of user stories.

We illustrate our approach on three complementary case studies. Through these case studies, we propose several Key Performance Indicators (KPIs) in order to assess the performance of our approach, and demonstrate its added value in automating the transition from user stories to models in Scrum. The remainder of the paper is structured as follow: Section 2 provides the background and the assumptions.Section 3 describes the proposed approach and Section 4 discusses the results as well as the evaluation of the approach. Section 5 raises the limitations and the threats to validity. Section 6 discusses the related works and finally, Section 7 concludes the paper.

2 BACKGROUND AND ASSUMPTIONS

In this section we discuss and clarify the key concepts underpinning this work as well as the considered assumptions.

2.1 User Stories and Use-Cases

A user story is a structured natural language description of requirements. It follows a compact template that describes the type of user, what they want and (optionally) why (Wautelet et al., 2014). User stories help to capture the description of the software features to be implemented from an end-user perspective.

Although many different templates exist, 70% of practitioners use the template (Lucassen et al., 2016a): "As a « type of user » , I want « goal », [so that « some reason »]". In our approach, we assume that the considered user stories rely on the following templates:

- *"As a « type of user », I want « goal », [so that/so « some reason »]".*
- *"As a « type of user », I'm able to « goal », [so that/so « some reason »]".*

Use-cases are on their side used to describe one specific interaction between the stakeholders and the system. For each user story, we consider that the « goal » part corresponds to the use-case.

However, the identified use-case can be either a whole use-case or a partial use-case, having inclusion or extension relationships to other use-cases (Sasse and Johnson, 1999). In our work, we assume that the « goal » part in each user story represents a whole single use-case.

2.2 Novel Approaches to Semantic Similarity

Semantic similarity plays a major role in various fields such as information retrieval, data integration and data mining (Varelas et al., 2005; Rodriguez and Egenhofer, 2003).

As we aim to group textual requirements in order to decompose the target system, we need to identify textual requirements that are semantically similar or related to one another.

Traditionally, semantic text matching have been defined using lexical matching and linguistic analysis (Lapata and Barzilay, 2005). Going beyond these traditional methods, finding semantic similarity between words is a fundamental part of text similarity. Then, it can be used as a primary stage to text similarity.

Word embedding allows to capture the context of a word in a document, semantic and syntactic similarity, as well as its relations with other words (Lebret and Collobert, 2014).

In this context, the vector-based approach of word2vec (Mikolov et al., 2013), a two-layer neural network for word embedding, enables predicting semantically similar words. As a prediction model, it has shown to outperform common traditional count models (Baroni et al., 2014).

Using the word2vec model, words with similar meaning end up lying close to each other. Thus, it gives best word representations. Moreover, it allows to use vector arithmetic to work with analogies, e.g., *vector (king) - vector (man) + vector (woman) = vector (queen).*

Although these new methods have not yet been applied in the requirements engineering literature, they are steadily being adopted in industry thanks to their excellent performance. Therefore, we rely on them in our approach to go from word-level to requirement-level semantic similarity.

2.3 Clustering

Textual requirements clustering refers to the process of taking a set of requirements and grouping them so that, requirements in the same cluster are similar and requirements in different clusters are different. In this context, we aim to adopt the clustering of user stories on the basis of their semantic similarity.

Clustering methods can be classified either as hierarchical or partitional (Hotho et al., 2005).
Hierarchical clustering algorithms work iteratively by joining, or dividing, clusters that are organized as a tree. These algorithms don't require any input parameters, they require only a similarity measure. Partitional clustering algorithms such as k-means require the number of clusters to start running. Thus, they rely heavily on the analyst's knowledge as they require the identification of the number of clusters to be generated in advance (Perner, 2002).

In our case, we don't have any condition on the number of clusters to be generated. Therefore, we use the hierarchical clustering methods, as they don't require us to pre-specify the number of clusters in advance. We employ the HAC algorithm (Zepeda-Mendoza and Resendis-Antonio, 2013). HAC algorithm works in a bottom-up manner, each object is initially considered as a single-element cluster (leaf). At each step of the algorithm, the two clusters that are the most similar are combined into a new bigger cluster (node). This procedure is iterated until all points are member of just one single big cluster. Moreover, using the HAC algorithm we can visualize a dendrogram, which represents the arrangement of the generated clusters in a hierarchical tree structure and it may help to graphically identify the hierarchy of the clusters.

In the next section, we explain in more details how we applied our clustering solution as well as how we generated UML use-cases from each cluster.

3 THE PROPOSED APPROACH

In this section, we present how our approach how our approach automatically generates UML models from natural language user stories. First, an overview of the process is given, then each subsection gives details on its implementation.

As shown in Figure 1, the user stories document is initially preprocessed. Text preprocessing is an essential step in the pipeline of a NLP system as this transforms the text into a form that is predictable and analyzable for machine learning tasks.

Then, we implement a semantic similarity module that computes the semantic similarity between each pair of the requirements. It takes as input the preprocessed requirements generated in the first step.

The similarity computation module includes two levels: (i) word level similarity, in which we use a pre-trained word embedding model, word2vec, to compute word-to-word semantic similarity as mentioned in Section 2.2, then we extend it to the (ii) requirement level similarity by means of the Mihalcea scoring formula for documents similarity computation (Mihalcea et al., 2006). The output of this module is a requirement similarity matrix.

Subsequently, in step 3, we feed the obtained requirements similarity matrix into the HAC algorithm.

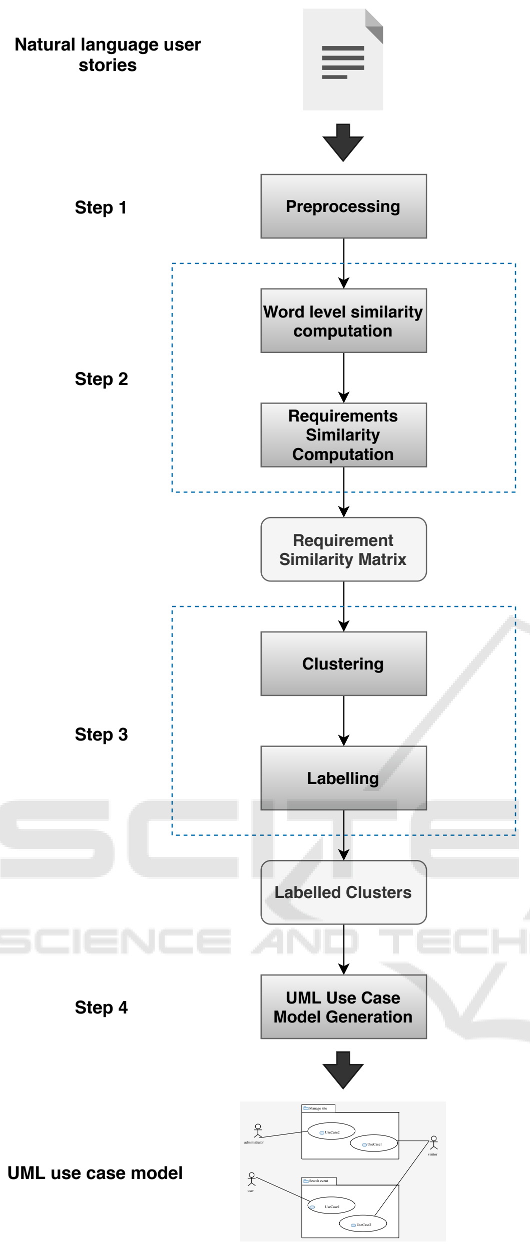

Natural language user stories
Step 1
Preprocessing
Step 2
Word level similarity computation
Requirements Similarity Computation
Requirement Similarity Matrix
Step 3
Clustering
Labelling
Labelled Clusters
Step 4
UML Use Case Model Generation
UML use case model

Figure 1: Architecture of the approach.

Step 4 illustrates the generation of UML use-cases from each identified sub-system. It takes as input the generated clusters of the semantically similar requirements from which we extract the relevant use case model elements using a set of specific NLP heuristics.
Finally, the extracted elements are programmatically mapped into UML use case model using the UML2 SDK of Eclipse.
More details on each of these steps are found in the following subsections.

3.1 Step 1: Preprocessing

This step is needed for transferring text from natural language to a machine-readable and analyzable format for further processing. Words should be transformed into numerical vectors that work with machine learning algorithms.

First, we start by normalizing requirements through four steps: (i) tokenization, i.e., the decomposition of a sentence into a set of individual words; (ii) stop-words removal, i.e., the elimination of common English words; (iii) punctuation removal ; (iv) stemming, i.e., the transformation of each word to its root (e.g: "adding" becomes "add").

Then, we use the word2vec model, a two-layer neural network that is used to produce word embeddings (i.e., vectors). Since a word embedding model is supposed to be of high quality when trained with large corpus, we use the pretrained word2vec model on 100 million words of Google News dataset[1].

However, even if the used corpus is too large (Google News), some domain-specific words did not exist in the corpus. In this case we assign to the missing word a random vector (Kim, 2014).

The resulting vectors are used for semantic similarity computation, which is explained in the next step.

3.2 Step 2: Semantic Similarity Computation Module

In this section, we explain in details how we computed the semantic similarity of each pair of requirement statements, taking into account both word level and requirement level.

- **Word Level Similarity.**
  At first level, we measure the semantic similarity between each pair of word vectors belonging to two different requirement statements using cosine similarity. The cosine similarity measure consists of computing the cosine of the angle between two words vectors. The cosine similarity of two similar words vectors is close to 1, and close to 0 otherwise.
- **Requirement Level Similarity.**
  Traditional approaches to find the semantic similarity between two text segments is to use lexical matching method, and produce a similarity score based on the number of lexical units that occur in both input segments. However, these lexical similarity methods cannot always identify the semantic similarity of texts (Mihalcea et al., 2006).

[1]https://code.google.com/archive/p/word2vec

Thus, going beyond the simple lexical matching methods used for this task, we were inspired by the work of Mihalcea et al. (Mihalcea et al., 2006), to derive the requirement level semantic similarity from the word level semantic similarity. Consequently, we used the Mihalcea scoring formula for text similarity computation (Equation 1).

$$Sim(R_1,R_2) = \frac{1}{2} \times \left(\frac{\sum_{w \in R_1} maxSim(w,R_2) \times idf(w)}{\sum_{w \in R_1} idf(w)} + \frac{\sum_{w \in R_2} maxSim(w,R_1) \times idf(w)}{\sum_{w \in R_2} idf(w)}\right) \quad (1)$$

First, we identify for each word w_1 in text requirement R_1, the word w_2 in the text requirement R_2 that have the highest semantic similarity $maxSim(w_1,R_2)$ (Equation 2), according to the word-to-word semantic similarity computation stated in the previous subsection. Next, the same process is applied to determine the most similar word in R_1 starting with words in R_2.

$$maxSim(w_1,R_2) = \max_{w_2 \in R_2} wordSim(w_1,w_2) \quad (2)$$

In addition to the similarity of words, we also take into account the specificity of words using the inverse document frequency idf. The specificity of a word w, $idf(w)$ (Equation 3) has been defined as the log of the total number of documents in the corpus divided by the total number of documents including that word (Jones, 1988).

$$idf(w) = \log(Total\ number\ of\ documents/\ Number\ of\ documents\ with\ word\ w\ in\ it) \quad (3)$$

The word similarities are then weighted with the corresponding word specificity. Thus, given two text requirement units R_1 and R_2, Mihalcea scoring formula combines word-to-word similarity and word specificity to derive text-to-text semantic similarity that is a potentially good indicator of the semantic similarity of two input texts.

The resulting requirement similarity has a value between 0 and 1, with a value of 1 indicating identical requirement text segments, and a value of 0 indicating no semantic overlap between the two segments. Therefore, given a document containing N requirements, the output of this step is a $N \times N$ semantic similarity matrix that contains the semantic similarity of each pair of requirements.

3.3 Step 3: Clustering Module

The semantic similarity matrix is fed into the HAC algorithm as stated in Section 2.3.

When the dendrogram is cut off at a specific point, a set of clusters is obtained which can represent a clustering solution of the system.

However, identifying the optimal number of clusters is not a trivial task. It can be either graphically identified by the analyst, then he can cut-off the dendrogram according to the desired number of the software sub-systems.

In order to reduce the need for manual intervention, we implement an operation that identifies automatically the best number of clusters using the Dunn index (Dunn, 1974) since a higher Dunn index indicates better clustering solution. Therefore, we compute the Dunn index each time, while varying the number of clusters.

The generated clusters are then labelled using the Gensim library for text summarization[2]. Gensim is a free Python library designed to automatically extract semantic topics from documents.

This summarizer is based on the ranks of text sentences using a variation of the TextRank algorithm proposed by Mihalcea et al. (Mihalcea and Tarau, 2004) and it summarizes a given text, by extracting one or more important sentences from the text. In a similar way, it can also extract keywords, i.e., the set of terms that best describe the document.

Thus, we label each generated cluster using Gensim's summarizer by extracting the best representative keywords embedded in the cluster.

3.4 Step 4: UML Use Case Model Generation Module

In this section we will explain how we automatically generate UML use case models from the resulting clusters that express early sub-systems. We define a set of specific NLP heuristics in order to extract use case model elements then, we programmatically create UML use case models on Eclipse Papyrus using the Eclipse UML2 tool SDK plugin.

NLP Heuristics. The extracted use case model elements are actors, use cases, and their associations. They are extracted using text chunking i.e., breaking the text down into smaller parts, with a set of specific NLP heuristics.

[2]https://radimrehurek.com/gensim/auto_examples/tutorials/run_summarization.html

We implement the proposed NLP heuristics using the Spacy library[3], a free and open-source library for advanced Natural Language Processing (NLP). The rules used to extract relevant elements from each user story, are the following:

- Actor is the first noun phrase chunk that exists after the pattern "as a". It can be a noun, noun plural or compound noun.
- Use case is the chunk that always exists between the patterns: "I'm able to" / "I want" and "so that/so" if it exists.
- In each user story, there is an association between the two elements resulting from the two previous rules.

UML2 Tool SDK Plugin to Automatically Generate Models. In order to illustrate our approach, we implemented a translator using the Eclipse UML2 SDK. The tool takes as input the extracted use case model elements, as well as the cluster to which they belong to. Then, it programmatically generates UML use case models.

The generated clusters are mapped into UML packages and each user story is mapped into a triplet actor, use case and their association.

4 EVALUATION

In order to demonstrate the applicability of our approach, we conduct three case studies. In this section, we present the Key Performance Indicators (KPIs) to be evaluated and the evaluation metrics. Moreover, we present the description of the case studies as well as the results.

4.1 Key Performance Indicators (KPIs)

- **KPI1: Accuracy of the Clustering Solution.**
  The aim of this KPI is to measure how much the identified clustering solution is accurate.
- **KPI2: Accuracy of the Generated Use Case Models.**
  For this KPI, we aim at identifying the accuracy of the use case models generated from each cluster.
- **KPI3: End-to-end Execution Time.**
  For this KPI, we aim to establish whether our approach runs within reasonable time in realistic settings.

[3]https://spacy.io/

In order to evaluate our approach w.r.t. the KPIs, we consider two metrics in our technique to be evaluated.

- **Clustering Evaluation Metric for KPI1.**
  In clustering theory, a wide range of metrics exist in order to evaluate the accuracy of clustering, each with its own advantages and limitations.
  We employ the Dunn index (Dunn, 1974) as an internal validity index. This internal index is used to evaluate the goodness of a clustering structure without reference to external information. It can be also used for estimating the number of clusters and the appropriate clustering algorithm without any external data. Subsequently, we compute this index for different number of clusters. The number of clusters for which we have the higher Dunn index value is the optimal number of cluster and indicates a better clustering solution.
  To the best of our knowledge, there is no baseline in the literature to compare our clustering results with. Consequently, we estimate whether a reasonable clustering is selected based on how well the clusters match the ground truth that is manually created by the team experts. To this end, we use precision, accuracy and F-measure metrics (Manning et al., 2005) to evaluate clustering results with respect to the manually identified clusters.
  True Positive (TP) elements are similar requirements assigned to the same cluster. False Positive (FP) elements are dissimilar requirements assigned to the same cluster. False Negative (FN) are similar requirements assigned to different clusters. Thus, the evaluation metrics are computed as follow:

  Precision = TP / (TP + FP)
  Recall = TP / (TP + FN)
  F-measure = 2 * Precision * Recall / (Precision + Recall)

- **UML Use Case Models Evaluation Metric for KPI2.**
  This evaluation is based on a comparison of the UML use case models, generated from our proposed approach, and the UML use case models manually created by experts from the user stories. We use the same metrics stated in clustering evaluation; precision, recall and F-measure (Manning et al., 2005), to evaluate the accuracy of the generated UML use case models.
  For each generated use case model, True Positive (TP) elements are the elements identified both manually and by the automatic approach. False Positive (FP) elements are the elements identified

by the automatic approach but not manually. False Negative (FN) are those identified manually but not by the automatic approach.

Furthermore, we compare our results with the approach proposed by Elallaoui et al. (Elallaoui et al., 2018). We consider the work in (Elallaoui et al., 2018) as our baseline because it closely relates to our work (extracting use case models from user stories) although authors didn't consider the clustering of user stories as a preliminary step towards model generation.

4.2 Description of Case Studies

We evaluated our approach on three real-world case studies available in the University of Bath's Institutional Repository (IRDUS1). The three case studies belong to different domains presenting different writing styles and different number of user stories. In the following, we introduce each in more detail:

- **CMS Company Case Study:**
  This case study involves a company developing complex Content Management System (CMS) products for large enterprises. 34 user stories are supplied. The user stories follow the template proposed by Cohn (Cohn, 2004) defined as follows **'As a [type of user], I want to [some goal] so that [some reason]'**. Despite the smaller size, this case study contains lengthy user stories with non-trivial sentence structuring.
- **Web Company Case Study:**
  This case study comes from a Dutch company that creates tailor-made web applications for businesses. The Web Company supplied 84 user stories covering the development of an entire web application focused on interactive story telling. The user stories in this case study rely on the template: **'As a [type of user], I'm able to [some goal] so that [some reason]'**.

  However, the structure of the user stories in this case study is more complex than in the CMS Company case study.
- **Archive Space Case Study:**
  Archive Space is an open-source software product created by archivists. The user stories of this project are available online[4]. It contains 51 user stories that rely on the template **'As a [type of user], I want [some goal]'**. All user stories in this collection omit the 'so that' token. Moreover, many user stories contain unnecessarily capitalized words, compound nouns, and idiosyncratic phrases such as **'...either the [creator | source | subject] of an [Accession | Resource | Resource Component]...'**

[4]archivesspace.atlassian.net

4.3 Results and Discussions

In this section, we present the results of applying our approach to the case studies.

4.3.1 Assessing KPI1: Accuracy of the Clustering Solution

For KPI1, the accuracy of the clustering solution depends on the choice of the number of clusters (i.e. clusters). We automated the identification of the optimal number of clusters by means of Dunn index as stated in Section 4.1. We run the HAC algorithm with different number of clusters for each run and we check Dunn index of the outcome. Thus, we carried out 6 runs for each case study. The best number of clusters is the number of clusters with the largest Dunn index value.

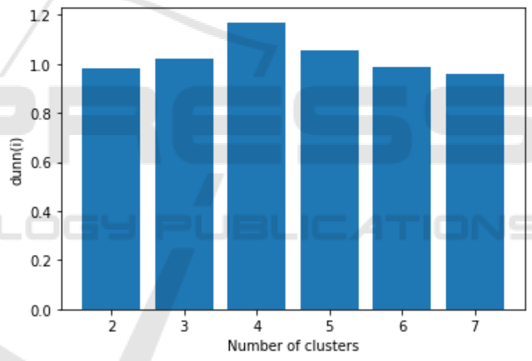


Figure 2: Dunn index bar graph of CMS Company case study.

For example, the best clustering result for the CMS Company case study was achieved in the fourth run, followed by the fifth, third and second ones. Thus, as shown in Figure 2 the optimal number of clusters for CMS Company case study is four clusters.

Moreover, it is possible to graphically verify the conformance of the distribution of the user stories and the optimal number of clusters that we identified based on the dendrograms as shown in Figure 3.

The output of this step is a set of labelled clusters of semantically similar user stories. Table 1 shows an example of a cluster labelled "page, URL, attribute" in the case of CMS Company case study which perfectly matches with the ground truth.

In Figure 4, we show results for precision, recall and F-measure metrics for the three case studies. Val-

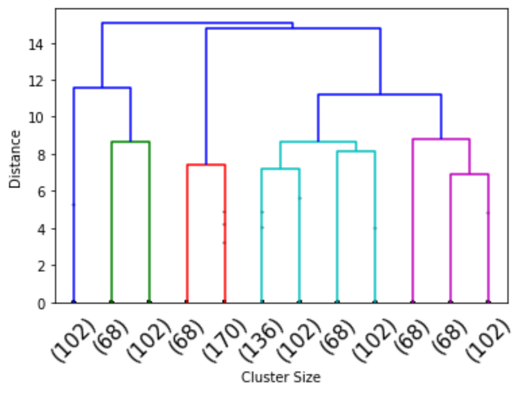


Figure 3: CMS Company case study dendrogram.

Table 1: Cluster: "page, URL, attribute".

1- As a marketeer, I want to set the title attribute of a link so I can improve the SE ranking of the website. 2- As a marketeer, I want to switch URLs when URL duplication occurs between pages so that I can solve conflicts between pages easily without having to search all pages in the tree. 3- As a marketeer, I want to create friendly URLs for my nested product pages so I can improve the SE ranking of the product section of my website. 4- As a marketeer, I want to set the rel attribute of external links so I can make sure that SE bots do no affect SE rankings for pages with many external links. 5- As a marketeer, I want to set canonical tags to individual pages so I can avoid duplicate content easily without having to set permanent redirects and thereby will improve the SE ranking of the website. 6- As a marketeer, I want to be solve URL conflicts immediately so I avoid not-friendly URLs and thereby will positively influence the overall SE ranking of the website. 7- As an editor, I want to assign page sections to pages using the current page section structure so I can easily and efficiently manage page sections assigned to pages.

ues are around 0.8 which is close to 1, and thus, indicate a relatively accurate clustering result. Moreover, the result reveals that the majority of the requirements is assigned to the correct cluster, which matches the ground truth accurately. Thus, the evaluation shows results with relatively high quality that can be applicable. Moreover, in spite of the fact that user stories of the three case studies belong to different domains, our experiments show that the clustering solution processed the same for the three case studies with the same parameter settings. What's more, we successfully avoid manual intervention by recommending an optimal number of clusters.

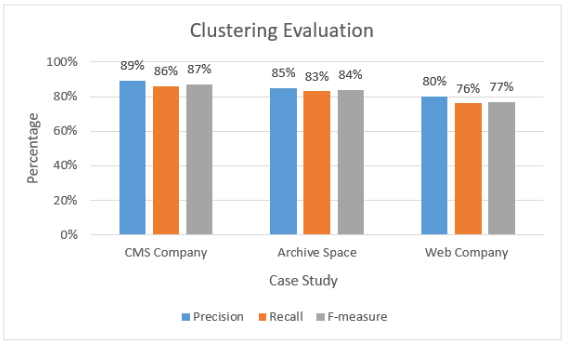


Figure 4: Bar graph of the clustering evaluation.

4.3.2 Assessing KPI2: Accuracy of the Generated Use Case Models

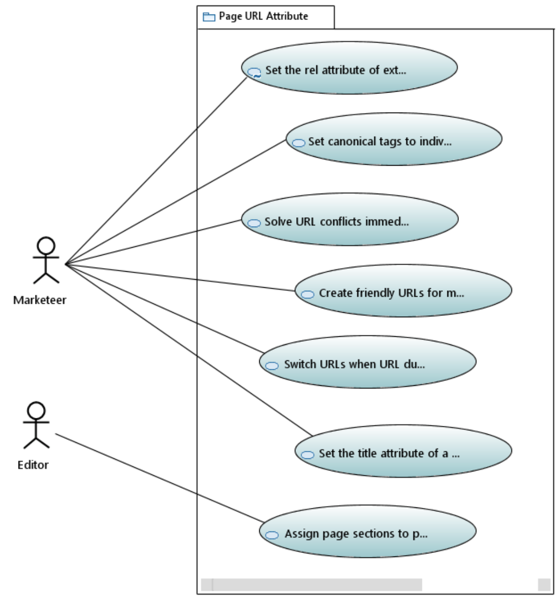
In order to evaluate KPI2, we specified True Positive (TP), False Negative (FN) and False Positive (FP) elements applied to actors, use cases and their relationships for each case study. Tables 3, 4 and 5 summarize the evaluation of the generated UML use case models in terms of precision, recall and F-measure.

Table 2: Example of the extracted elements for the cluster "page URL attribute".

Actor	Use case
marketeer	set the title attribute of a link
marketeer	switch URLs when URL duplication occurs between pages
marketeer	create friendly URLs for my nested product pages
marketeer	set the rel attribute of external links
marketeer	set canonical tags to individual pages
marketeer	solve URL conflicts immediately
editor	assign page sections to pages using the current page section structure

Table 3: Accuracy of the generated UML use case model for the "Web Company" case study.

	Actors	Use Cases	Relation-ships
	TP FP FN	TP FP FN	TP FP FN
	84 0 0	84 2 7	69 17 10
Precision	100%	97%	97%
Recall	100%	92%	90%
F-measure	100%	94%	93%



Page URL Attribute
Set the rel attribute of ext...
Set canonical tags to indiv...
Solve URL conflicts immed...
Create friendly URLs for m...
Switch URLs when URL du...
Set the title attribute of a ...
Assign page sections to p...
Marketeer
Editor

Figure 5: Overview of the generated UML use case model for the "CMS Company" case study.

Table 4: Accuracy of the generated UML use case model for the "CMS Company" case study.

	Actors	Use Cases	Relation-ships
	TP FP FN	TP FP FN	TP FP FN
	34 0 0	32 2 2	28 4 4
Precision	100%	94%	88%
Recall	100%	94%	88%
F-measure	100%	94%	88%

Table 5: Accuracy of the generated UML use case model for the "Archive Space" case study.

	Actors	Use Cases	Relation-ships
	TP FP FN	TP FP FN	TP FP FN
	56 0 0	51 1 6	33 11 6
Precision	100%	98%	75%
Recall	100%	89%	84%
F-measure	100%	93%	79%

The accuracy of the generated UML use case models is reasonable for the three case studies.

For actors, F-measure value is equal to 100% as we succeeded to extract all the actors even actors with compound nouns such as "read Only user" and "repository Manager".

For use cases and relationships, F-measure values take high-range (93%-94%) and (79%-93%) respectively. In relationships detection, these values are due in particular to inclusion or extension relationships between use-cases that are not supported by our approach.

Moreover, we compare our results to the work in the baseline (Elallaoui et al., 2018) which has been evaluated only for the Web Company case study. Results indicate that our approach succeeded in extracting all the actors while the approach in the baseline (Elallaoui et al., 2018) failed to extract actors with compound nouns such as "newly registered user". Moreover, for use-cases and relationships, our approach achieves better F-measure results than the baseline (Elallaoui et al., 2018) by 8 and 10 percentage points respectively.

As an illustration, Figure 5 shows the generated UML use case model based on the extracted elements presented in Table 2 for the cluster "page, URL, attribute" (Table 1).

4.3.3 Assessing KPI3: Execution Time

For KPI3, table 6 shows the execution time for the different steps of our approach measured on a laptop with a 2.10 Ghz Intel (R) Core (TM) i7-4600U CPU and a 8GB of memory.

Table 6: Execution time.

Case Study	Phase	Execution Time
CMS Company	preprocessing	5s
	Clustering (similarity computation+HAC +labelling)	28s
	UML use case generation	2s
	Total	35s
Web Company	preprocessing	11s
	Clustering (similarity computation +HAC +labelling)	56s
	UML use case generation	5s
	Total	1m12s
Archive Space	preprocessing	7s
	Clustering (similarity computation +HAC +labelling)	35s
	UML use case generation	3s
	Total	45s

For the three case studies, the execution time is dominated by the clustering step as well as the inferring of the words vectors in the semantic similarity computation step. Figure 6 shows a linear growth trend for the impact of the number of user stories on the execution time. Given such linear relation and the fact that the end-to-end execution time takes few seconds to few minutes, we anticipate that our approach should be practical for much larger user stories documents.

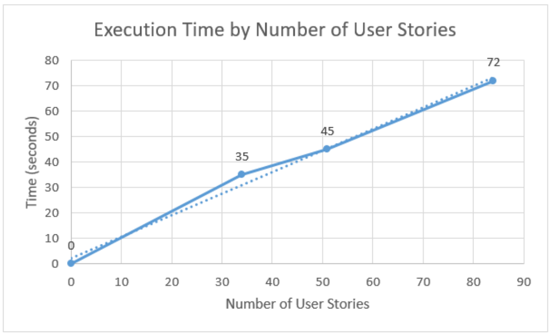


Figure 6: Impact of the number of user stories on the execution time.

5 THREATS TO VALIDITY

In this section, we discuss the limitations of our proposal in terms of internal threats, construct threats and external threats. These threats are as follows:

Internal Validity. With regard to computing word similarity, some domain-specific words don't occur in the corpus used to train the word vector space which might slightly affect the efficiency of word similarity computation. To mitigate this limitation, we map such words to a random vector.

Construct Validity. With regard to assessing the accuracy of the clustering, the definition of what a "better" cluster is might be a subjective procedure since there is no baseline in the literature to compare our clustering results with. Although this may lead to bias of evaluating clusters, we still achieve relatively accurate results compared with ground truth.

External Validity. We evaluate the applicability of our approach on three case studies. Although the evaluation gives promising results, we need to evaluate the approach on further case studies including industrial case studies.

Ultimately, as mentioned above, there are several threats to our approach that have to be considered in order to improve our proposal.

6 RELATED WORK

Many work found in the literature aim at deriving models from early requirements expressed in natural language.

Fabian et al. (Gilson and Irwin, 2018) propose an automatic transformation from user stories to robustness diagrams. The approach aims at helping requirements engineers and users to validate user stories and to perform structured analysis. Thus, the work consists in identifying loosely defined or ambiguous stories, extracting core concepts as well as relations between them using NLP techniques.

Ellaoui et al. (Elallaoui et al., 2015), propose an approach that helps engineers to reduce ambiguity in requirements specifications in scrum processes. The approach consists in generating sequence diagrams from user stories in order to generate test cases. It is based on a set of rules that are used to extract model elements. Then, an XMI file for each user story defining the corresponding sequence diagram is generated.

In (Elallaoui et al., 2018), the authors propose an automatic transformation of user stories into UML use case diagrams to assist the work of the development team and the Product Owner. The approach is based on a set of NLP heuristics that allows to extract use-case elements. However, the proposed approach is unable to detect actors with compound nouns as well as inclusion and extension relations between use-cases.

In a similar approach, Arora et al. (Arora et al., 2016) developed a domain model extractor from natural language requirements. They are able to identify classes, relations and attributes with a variable success rate.

These works commonly employ a syntactic transformation to bridge the gap between natural language requirements and models. Although there are multiple approaches that transform requirements to models, decomposing the target system into a set of subsystems from early requirements is also needed as the number of requirements grows. Hence, the usage of clustering techniques in the early phases of software engineering has gained a lot of attention in recent years.

In (Amannejad et al., 2014), the authors developed a tool based on hierarchical clustering of requirements in order to propose a packaging solution for software engineers. They defined a similarity measure that aims to cluster classes with high number communication in the same package. However, the optimal number of clusters is manually selected by software engineers based on the hierarchical tree generated by the clustering algorithm.

Lucassen et al. (Lucassen et al., 2016b) present an approach based on concepts clustering to visualize requirements at different levels of granularity. They employed word2vec as a prediction model to compute similarity between concepts.

Agustin et al. (Casamayor et al., 2012) propose an initial clustering of responsibilities from requirements, in order to detect architecture components. The approach is validated using four different clustering algorithms and several validity metrics. The similarity function is computed according to the verb phrase each responsibility contains and the direct object it is related to.

Barbosa et al. (Barbosa et al., 2015) present an approach to cluster and sequence user stories in order to assist software engineers in the implementation phase. They employed a clustering algorithm and the silhouette score to identify the best clustering solution.

In (Jalab and Kasirun, 2014), the authors propose an approach that clusters similar requirements in order to reuse them in software product lines (SPLs). They compared the performance of two clustering algorithms based on a distance measure in order to identify similar requirements.

In (Salman et al., 2018), the authors demonstrate the use of the HAC algorithm to break-down the project into a set of sub-projects based on related functional software requirements. They use traditional vector space models to vectorize text requirements. The clustering framework that they propose is dynamic and can be extended to include different clustering algorithms and distance measures.

All these techniques inspire our work. However, some of these approaches suffer from a lack of automation when defining the optimal number of clusters (Amannejad et al., 2014), others rely on the similarity between words or concepts in each requirement (Lucassen et al., 2016b), (Casamayor et al., 2012).

Moreover, some of them utilize traditional count models, for instance, Vector Space Model (VSM) (Salman et al., 2018), Latent Semantic Analysis (LSA) (Jalab and Kasirun, 2014), and TF-IDF (Barbosa et al., 2015) to calculate the similarity. Meanwhile these count models can not achieve synonymy and polysemy, which decreases the similarity accuracy. Consequently, they usually achieve worse results than prediction models (Baroni et al., 2014).

The main novelty of our proposal is that we benefit from using word2vec as prediction model, to compute word level similarity and then, derive the requirement level similarity using a scoring formula for text similarity. Then, we automatically generate clusters from natural language user stories using HAC algorithm and we implement an operation that recommends the optimal number of clusters in order to avoid the manual intervention. Finally, we implement a set of specific NLP heuristics to extract relevant use case model elements in order to instantiate our approach in UML use case models and hence, foster the integration of model-based approaches in Scrum process.

7 CONCLUSION

In this paper, we proposed a machine learning based approach to automatically derive preliminary UML behavioral models from early natural language user stories in Scrum process. The core of the approach is the clustering of the natural language requirements with similar functionalities in order to break-down the system into sub-systems.

Such automatic decomposition of the software system is highly desired at early stages specifically for large scale systems to facilitate the design process and help in saving in cost and time.

Therefore, we employed the HAC algorithm to group the semantically similar user stories. In order to improve the accuracy of the generated clusters, semantic similarity was computed taking into account both word level and requirement level similarity. Word level similarity was firstly computed using word2vec as prediction model, then it was extended to the requirements level using the Mihalcea scoring formula for text similarity computation. Then, we implemented a set of specific NLP heuristics to extract relevant use case model elements from each cluster. Subsequently, our approach was illustrated by the generation of sub-systems expressed as UML use case models and hence, it promotes the integration of model-based methodologies in Scrum.

Our approach was evaluated through three case studies and a set of KPIs. For the clustering solution, we succeeded to avoid manual intervention by recommending an optimal number of clusters. Furthermore, evaluation results of the generated clusters against ground truth clusters made by experts reveals reasonable F-measure values which are around 80%. As for precision and recall values of the generated UML use case models, they are equal to 100% for actors detection, up to 98% for use-cases detection and up to 97% for relationships detection. Meanwhile,for the same case study presented in the baseline (Elallaoui et al., 2018), we achieved a better F-measure score. Our algorithm succeeded to detect all the actors compared to (Elallaoui et al., 2018) and F-measure values of use-cases and relationships detection, are better by 8% and 10% respectively.

Given these promising results, we plan to extend our work to support other templates of user stories. Moreover, we plan to further assess the effectiveness of the approach on larger software requirement documents, possibly taken from different domains using a prediction model (word2vec) trained with domain-specific corpus.

REFERENCES

Alexander, I. F. and Maiden, N. A. M. (2004). Scenarios,stories, use cases: Through the systems development life-cycle.

Amannejad, Y., Moshirpour, M., Far, B. H., and Alhajj, R. (2014). From requirements to software design: An automated solution for packaging software classes. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, pages 36–43.

Arora, C., Sabetzadeh, M., Briand, L. C., and Zimmer, F. (2016). Extracting domain models from natural-language requirements: approach and industrial evaluation. In *MODELS '16*.

Barbosa, R., Januario, D., Silva, A. E., de Oliveira Moraes, R. L., and Martins, P. (2015). An approach to clustering and sequencing of textual requirements. *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, pages 39–44.

Baroni, M., Dinu, G., and Kruszewski, G. (2014). Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 238–247, Baltimore, Maryland. Association for Computational Linguistics.

Bittner, K. (2002). *Use Case Modeling*. Addison-Wesley Longman Publishing Co., Inc., USA.

Booch, G., Rumbaugh, J. E., Jacobson, I., and Hoberman, S. (2015). The unified modeling language user guide (2nd edition).

Casamayor, A., Godoy, D., and Campo, M. R. (2012). Functional grouping of natural language requirements for assistance in architectural software design. *Knowl. Based Syst.*, 30:78–86.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison Wesley Longman Publishing Co., Inc., USA.

Dunn, J. C. (1974). Well-separated clusters and optimal fuzzy partitions.

Elallaoui, M., Nafil, K., and Touahni, R. (2015). Automatic generation of uml sequence diagrams from user stories in scrum process. In *2015 10th International Conference on Intelligent Systems: Theories and Applications (SITA)*, pages 1–6.

Elallaoui, M., Nafil, K., and Touahni, R. (2018). Automatic transformation of user stories into uml use case diagrams using nlp techniques. In *ANT/SEIT*.

Gilson, F. and Irwin, C. (2018). From user stories to use case scenarios towards a generative approach. *2018 25th Australasian Software Engineering Conference (ASWEC)*, pages 61–65.

Hotho, A., Nürnberger, A., and Paass, G. (2005). A brief survey of text mining. *LDV Forum*, 20:19–62.

Jalab, H. A. and Kasirun, Z. M. (2014). Towards requirements reuse: Identifying similar requirements with latent semantic analysis and clustering algorithms.

Jones, K. S. (1988). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 60:493–502.

Kassab, M. (2015). The changing landscape of requirements engineering practices over the past decade. In *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 1–8.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *EMNLP*.

Lapata, M. and Barzilay, R. (2005). Automatic evaluation of text coherence: Models and representations. In *IJCAI*.

Lebret, R. and Collobert, R. (2014). Word embeddings through hellinger pca. In *EACL*.

Lindvall, M., Basili, V. R., Boehm, B. W., Costa, P., Dangle, K. C., Shull, F., Tvedt, R. T., Williams, L. A., and Zelkowitz, M. V. (2002). Empirical findings in agile methods. In *XP/Agile Universe*.

Löffler, R., Güldali, B., and Geisen, S. (2010). Towards model-based acceptance testing for scrum. *Softwaretechnik-Trends*, 30.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E. M., and Brinkkemper, S. (2016a). The use and effectiveness of user stories in practice. In *REFSQ*.

Lucassen, G., Dalpiaz, F., van der Werf, J. M. E. M., and Brinkkemper, S. (2016b). Visualizing user story requirements at multiple granularity levels via semantic relatedness. In *ER*.

Manning, C. D., Raghavan, P., and Schütze, H. (2005). Introduction to information retrieval.

Mihalcea, R., Corley, C., and Strapparava, C. (2006). Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*.

Mihalcea, R. and Tarau, P. (2004). Textrank: Bringing order into texts. In *EMNLP 2004*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *ArXiv*, abs/1310.4546.

Perner, P. (2002). Data mining - concepts and techniques. *KI*, 16:77.

Rodriguez, M. A. and Egenhofer, M. J. (2003). Determining semantic similarity among entity classes from different ontologies. *IEEE Trans. Knowl. Data Eng.*, 15:442–456.

Salman, H. E., Hammad, M., Seriai, A.-D., and Al-Sbou, A. (2018). Semantic clustering of functional requirements using agglomerative hierarchical clustering. *Information*, 9:222.

Sasse, M. A. and Johnson, C. (1999). Human-computer interaction interact '99: Ifip tc. 13 - 1999 edinburgh.

Schwaber, K. (1997). Scrum development process.

Varelas, G., Voutsakis, E., Raftopoulou, P., Petrakis, E. G. M., and Milios, E. E. (2005). Semantic similarity methods in wordnet and their application to information retrieval on the web. In *WIDM '05*.

Verner, J. M., Cox, K., Bleistein, S. J., and Cerpa, N. (2005). Requirements engineering and software project success: an industrial survey in australia and the u.s. *Australasian J. of Inf. Systems*, 13.

Wautelet, Y., Heng, S., Kolp, M., and Mirbel, I. (2014). Unifying and extending user story models. In *CAiSE*.

Zepeda-Mendoza, M. L. and Resendis-Antonio, O. (2013). *Hierarchical Agglomerative Clustering*, pages 886–887. Springer New York, New York, NY.