ZAMBIA INFORMATION COMMUNICATION TECHNOLOGY (ICT) JOURNAL

Volume 6 (Issue 1) (2022) Pages 1-6

Exploring the Decomposition of Epics using Natural Language Processing

Bokang Seitlheko[a] and Laban Mwansa[b]

Department of Electrical, Electronics and Computer Systems Engineering
Cape Peninsula University of Technology
Cape town, South Africa
a. bokanggeorge@gmail.com b. lmmwansa@yahoo.com

***Abstract*—Agile user requirements are typically givens as user stories written using natural language and they come in different forms. The most complex form of stories to work with are epics. If epics are poorly understood, they can contribute to threats regarding the sprints or projects becoming behind schedule. It can be attributed to the epic's complexity. The research aimed to explore and attempt the use of Stanza from the Stanford NLP group in the decomposition of epics by creating a text generative model. We have also utilised the chunking technique to formulate the tasks from the generated user stories by identifying the linguistic structure through the aid of a POS tagger. The obtained results illustrate that the stanza can be utilised in the requirements engineering domain such as Sprint backlog grooming. The benefits of this research work are enormous considering that sprint backlog grooming takes considerable time and is always in iterative mode. Agile teams will also benefit from this work by efficiently using sprint timeboxes with minimal sprint planning effort. This will enable agile teams to spend more time delivering the right solutions with reduced sprint planning time and effort.**

Keywords—epic story refinement, natural language processing, user stories, Scrum

I. INTRODUCTION

Over the last decade, user stories have been ruled as the representation of user requirements in agile software development (ASD). They are described as the semi-structured and concise representation of user requirements transcribed using natural language (NL) [1]. These stories are transcribed concisely to enable the fast progression of software development while maximising business value. Being concise ascribe to the flexibility and adoption of this notation in a dynamic environment such as ASD where just enough documentation is mandatory.

The widely embraced user stories template by practitioners is given as follows: As < actor >, I want to < action > so that < business value or reason > is easy to comprehend and employ. Albeit easy to use, there are few complications. Mostly, large stories do not fit the Sprint, and this brings about negative repercussions on the projects; the sprint becomes partially complete during epics development and stimulates schedule overrun. These large stories are referred to as epics. To address this challenge, the preliminary rule of thumb is to decompose or refine these epics into small manageable stories. Decomposing is the reduction of an epic or large story into small manageable stories such that it can fit the sprint.

It is essential to decompose epics since the accuracy of the sprint planning lies in the heart of the user story's (US) complexity and inherent risk. Small user stories give the development team (DT) the confidence to select them over epics during sprint planning because there are no unanticipated emergent details. Additionally, small stories bring about adequate architecture, and their efforts are easy to estimate. Furthermore, the decomposed stories have a higher probability of being completed on time during their execution than the larger stories. However, there is still a challenge to automate the refinement of epics into user stories in an agile environment, especially in the context of Scrum methodology.

Other researchers rely on the use of traditional approaches in Agile methodologies. Traditional approaches require manual human intervention and expertise to operate. The use of two traditional approaches horizontal and vertical slicing in Agile methodology is still acceptable. However, these techniques suffer from scalability problems when applied to extensive projects with large requirements.

This paper draws inspiration from the prior success of applying NLP on a diverse array of automating the generation of Agile artefacts. This includes the transformation of user stories into use cases [2], the construction of test cases from software requirements specification (SRS) documentation expressed as natural language by [3], the automatic generation of user acceptance test cases from a document comprised of use case specification by Wang *et al*, etc. Consequently, this paper attempts to harness the power of spacy-stanza to decompose stories. The Stanza Part of Speech (POS) tagger was used to analyse the linguistic structure of the generated user stories to form tasks. The process of extracting tasks from generated stories was accomplished using chunking with the aid of POS tagging.

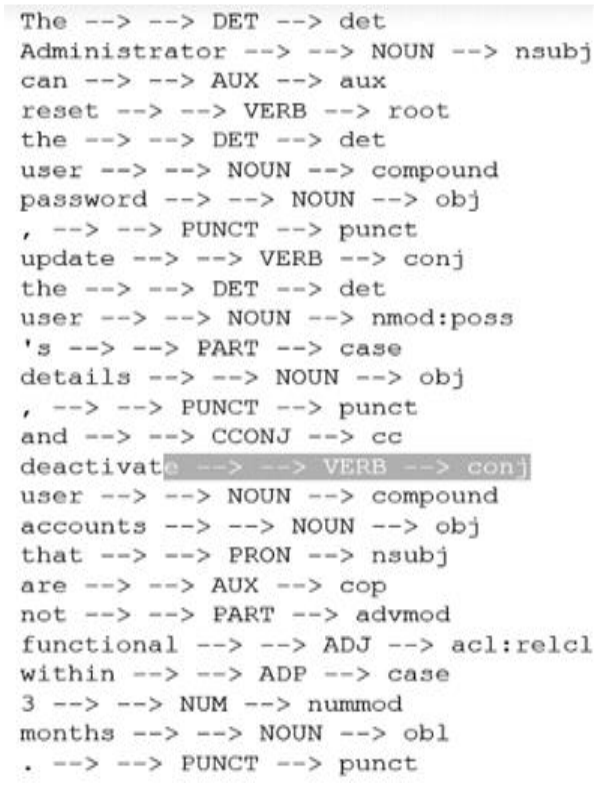
This paper is divided into four subsections. In section 2, the description of the background study is described to familiarize the reader with the technology and the terms used. Furthermore, section 3 describes the indulged methodology to attain the objective of the research, while four is the results and the discussion of the attained results. Lastly, we conclude the use of NLP in generating new user stories and do the recommendations.

II. BACKGROUND

A. *Baseline: conceptual anatomy of Agile epic stories*

Massive user stories are sometimes referred to as epics [5][6]. Usually, this type of story consists of two or more action verbs based on their analyzed linguistic structures. For example, consider the following epic:

1) The Administrator can reset the user password, update the user's details, and deactivate user accounts that are not functional within 3 months.



```
The --> --> DET --> det
Administrator --> --> NOUN --> nsubj
can --> --> AUX --> aux
reset --> --> VERB --> root
the --> --> DET --> det
user --> --> NOUN --> compound
password --> --> NOUN --> obj
, --> --> PUNCT --> punct
update --> --> VERB --> conj
the --> --> DET --> det
user --> --> NOUN --> nmod:poss
's --> --> PART --> case
details --> --> NOUN --> obj
, --> --> PUNCT --> punct
and --> --> CCONJ --> cc
deactivate --> --> VERB --> conj
user --> --> NOUN --> compound
accounts --> --> NOUN --> obj
that --> --> PRON --> nsubj
are --> --> AUX --> cop
not --> --> PART --> advmod
functional --> --> ADJ --> acl:relcl
within --> --> ADP --> case
3 --> --> NUM --> nummod
months --> --> NOUN --> obl
. --> --> PUNCT --> punct
```

Fig. 1. Analysed linguistic structure of an epic requirement.

These stories are massive to manage on a single sprint, so they are moved on to the next sprint to avoid overfitting. Overfitting is when the user story is too large and cannot be coupled with other user stories very well due to its capacity. It is crucial to note that user stories should not be too small or too big in terms of estimates. If the estimations of the user's story points are too small, say 0.3, there is a possibility of facing micromanagement. Moreover, a 60 points story stands a chance or high risk of ending up being a partially complete sprint [6].

There are two methods of splitting user stories into manageable tasks: horizontal and vertical slices. A more recent study suggested that the utilization of predictive analysis can be deployed in managing user stories [7].

B. *Related work*

This section discusses efforts made to address the decomposition of agile epics starting from traditional approaches (horizontal and vertical approaches) to contemporary state of art techniques such as NLP.

1) *Horizontal and vertical*

According to [8] the widely practised decomposing technique within the boundaries of Agile is US mapping. It describes the decomposition of large USs from the user's perspective; It provides the highest level of requirements abstraction. In story mapping, large stories are coarse-grained from epics to stories until their constituent's tasks. For instance, "create registration form" and "create a login page for the system" are good examples of high-level requirements. In addition, they further explored how different agile methodologies such (as XP, Scrum, and Scrum with Kanban) engage in the decomposition process. Their research results revealed that the utilization of traditional processes is still applied in the agile process during the splitting of stories. Moreover, the most proficient method among the discussed methods is Scrum with kanban followed by XP. The success of Scum with Kanban was due to the use of the vertical slicing technique. Albeit its popularity among Agilist, story mapping is achieved by human expertise.

Vertical slicing is the technique of decomposing an epic by touching aspects of every layer such as from the User Interface (UI) to the database. It encourages the delivery of product increments frequently to the end-users such that they provide feedback and incorporate updates within the subsequent iteration. In a study contacted by [9], four teams were deployed to study and determine how efficient is horizontal and vertical slicing in US decomposition. Vertical tends to have more positive traits than horizontal slicing in terms of risk and completion of the project. The utilization of the horizontal technique parades no functionality to the end-users but partially completed tasks which leads to reiteration and delivers ineffective sprints.

Other studies are worth noting. Lawrence proposed a strategy that decomposes epics through the reprioritization and isolation of requirements. To reach their goal, they decompose epics into small fragments and discarded stories with no value or importance. This technique was found to improve the isolation and decreases the inter-dependencies between user stories.

2) *Contemporary state of art*

The degree of automating requirements decomposition is scarce in agile software engineering in both the academic world and enterprise environment. However, the contemporary state of the art only provides the roadmap of possibilities of using NLP to address the challenges faced by manual techniques. The preliminary attempt which paved the way for the decomposition of requirements using NLP was recently shed to light by [10]. They studied linguistic structures that characterize USs together with their corresponding sprint backlog items. To achieve this, they used the Stanford Part-of-Speech (POS) tagger to determine the structure of the task labels. POS tagger is especially used in NLP to extract language structure such as verbs, adjectives, nouns, and others. Their results revealed the useful insights that can be employed to form the linguistic structure of tasks

III. Proposed Solution

We propose the utilization of the spacy-stanza library to generate the user stories and tasks from a file comprised of Agile epics. Figure 2 shows the proposed method divided into two pipelines to enhance the tool’s performance.

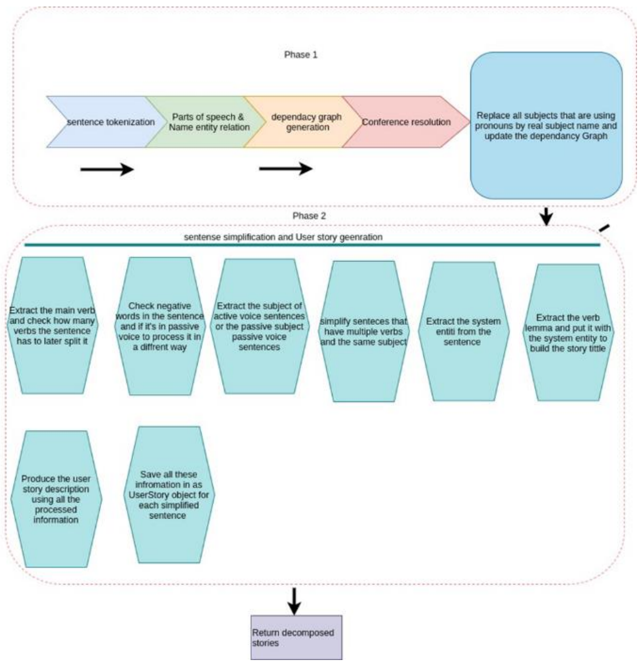

Phase 1
sentence tokenization
Parts of speech & Name entity relation
dependacy graph generation
Conference resolution
Replace all subjects that are using pronouns by real subject name and update the dependancy Graph
Phase 2
sentense simplification and User story geenration
Extract the main verb and check how many verbs the sentence has to later split it
Check negative words in the sentence and if it's in passive voice to process it in a diffrent way
Extract the subject of active voice sentences or the passive subject passive voice sentences
simplify senteces that have multiple verbs and the same subject
Extract the system entiti from the sentence
Extract the verb lemma and put it with the system entity to build the story tittle
Produce the user story description using all the processed information
Save all these infromation in as UserStory object for each simplified sentence
Return decomposed stories

Fig. 2. Outline the design process of decomposing epics
Adapted from [11]

A. *First pipeline*

There are six processes executed in this stage; sentence segmentation, Part of Speech (POS), Named Entity Relation (NER), dependency graph generation, conference resolution and replacing the subject mentions from the text by identifying its pronouns [11]. This paper has adopted the use of new python NLP libraries called Stanza and Spacy-Stanza. The first four tasks were resolved by utilizing spacy-stanza pipelines and annotations which returned the POS tag of words, lemma and dependency graph of text provided as an output. Dependency graph returns a syntactic relationship between words in a sentence. The graph can consist of POS tags, root, etc. To visualize the dependencies, we imported the displacy library from spacy.

The subsequent step resolved conference resolution between the sentences by utilizing Stanza CoreNLPClient interface annotators. This process simply replaces the pronouns with their correlated subject names or noun present in the Mentions. If similar subjects refer to the same pro/noun in the text given, the algorithm returns none and continues with the output of the current dependency graph where the graph's metadata acts as the input to the second pipeline. For demonstration purposes, we have illustrated the process of attaining the first output pipeline by using the text below.

Sentence 1:

1) *"*The bank Administrator views customer profile. But he cannot delete transactions history.*"*

After the text went through all processes from the first pipeline, the output comes as modified text below:

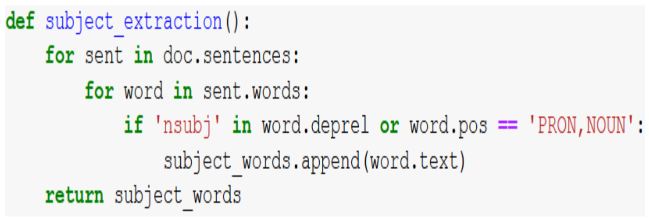
Transformed sentence:

2) The bank Administrator views the customer profile. But the bank Administrator cannot delete transaction history.

It is worth noting that the algorithm identified The bank administrator and he as the subjects in both sentences. However, the pronoun "he" refers to the same subject as "the bank administrator", therefore the algorithm suggested the replacement of pronouns with their respective subjects. Therefore, this triggers the dependency graph to be updated and return the modified text.

B. *Second pipeline (Phase 2)*

The second pipeline receives metadata from the first pipeline's output and executes the most fundamental natural language tasks to decompose the epics into user stories. Having the input as a dependency graph from the first pipeline permits us to perform text analysis and extract essential information to generate user story information. For instance, to extract the user/actor of the user story from the dependency graph, we have implemented a function that facilitates the extraction of the subject representatives of nouns and returns them as a list. This was accomplished by extracting a word with POS tags NOUN, PROPN and dependency token either nsubj or nsubjpass, compound. The code illustrates how to extract the user/actor for the user story generation.



```
def subject_extraction():
    for sent in doc.sentences:
        for word in sent.words:
            if 'nsubj' in word.deprel or word.pos == 'PRON,NOUN':
                subject_words.append(word.text)
    return subject_words
```

Fig. 3. Code snippet for extracting an actor

The first line from the code performs sentence segmentation while the second and third lines perform tokenization. The fourth line searches all the words that are subjects with active voice (nsubj) by using the dependency attribute deprel. The items from this list are retrieved later to form part of the user story's information, user /actor. For instance, the template that this paper followed is given as:
Template: "As "+ <user/actor> + "I want to be able to "+ <phrase>**.**

C. *Generating user stories*

To generate the user story, the algorithm starts by counting the number of verbs in each sentence as indicated in the activity diagram in figure 4. The processing is given in two categories, single verbs and multiple verbs. For each category, there are different steps to follow until the user stories and tasks are generated.

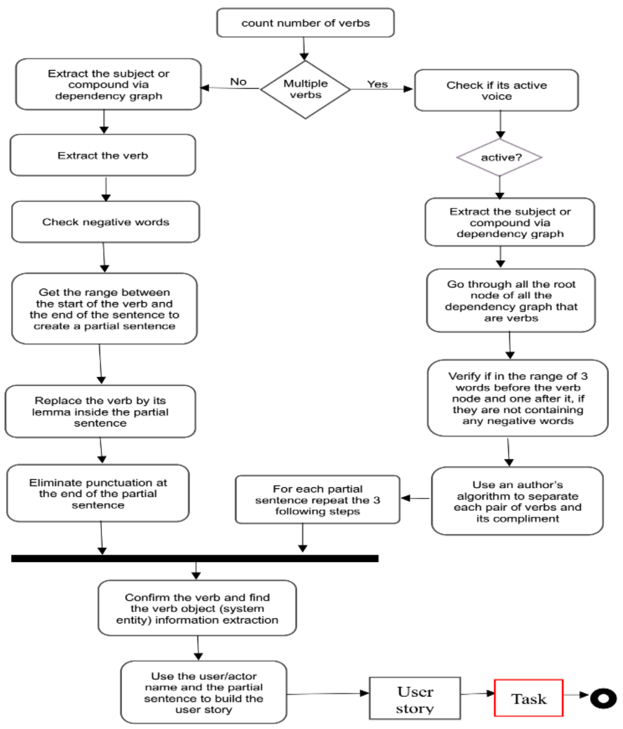

count number of verbs
Multiple verbs
No
Yes
Extract the subject or compound via dependency graph
Extract the verb
Check negative words
Get the range between the start of the verb and the end of the sentence to create a partial sentence
Replace the verb by its lemma inside the partial sentence
Eliminate punctuation at the end of the partial sentence
Check if its active voice
active?
Extract the subject or compound via dependency graph
Go through all the root node of all the dependency graph that are verbs
Verify if in the range of 3 words before the verb node and one after it, if they are not containing any negative words
Use an author's algorithm to separate each pair of verbs and its compliment
For each partial sentence repeat the 3 following steps
Confirm the verb and find the verb object (system entity) information extraction
Use the user/actor name and the partial sentence to build the user story
User story
Task

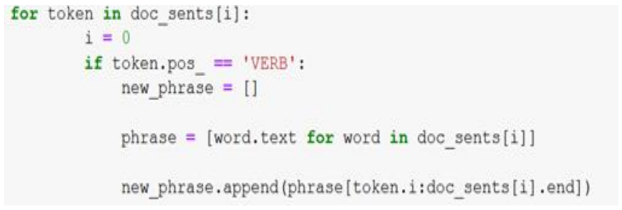
Fig. 4. Activity diagram decomposes user story and task

1) Algorithm for a single verb

The process starts by counting the number of verbs present in each sentence. Most algorithm steps followed in the activity diagram in figure 4.1 are self-explanatory. After that, we find the subject from the sentence using the dependency graph by finding the keyword key nsubj. The nsubj denotes that the sentence is in active voice, and it is a nominal subject, while nsubjpass denotes that the sentence it is in passive mode. However, this paper focuses on text written in active mode. We, therefore, discard all text written in passive mode. For demonstration purposes, we will continue with the first sentence from the first pipeline.

Sentence 1: *The bank Administrator views the customer's profile*

Following the algorithm proposed by [11], iterate through the entire dependency graph and count the number of verbs in each sentence by finding their POS tag with the keyword VERB. It is worth noting that this research only implemented the user stories that are inactive mode. The activity diagram in figure 4 illustrates the algorithm used to attain user stories and tasks. From the sentence, the verb found was **views.** Find the position of the identified verb and start the partial sentence generation from the verb's index until the end sentence. Save this information in a string variable called a partial phrase. Figure 5 illustrates the code snippet that this paper has developed to identify the verbs' index and generate a partial sentence.


```
for token in doc_sents[i]:
        i = 0
        if token.pos_ == 'VERB':
            new_phrase = []

            phrase = [word.text for word in doc_sents[i]]

            new_phrase.append(phrase[token.i:doc_sents[i].end])
```

Fig. 5. Code snipped for partial sentence generation

The first line of the code determines tokens found in a sentence. The doc_sent[i] in this case indicates the specific sentence we are focusing on with a single verb. Line 3 identifies the verb from the sentence by using POS tag VERB. After finding the position of the verb in a sentence, get the index of the verb by using the keyword **token.i**. The next task was to determine the end of the sentence. As indicated in line 6, the end of the sentence was found by using the keyword **end** with the sentence. After finding the position of the verb and the end of the sentence, construct a partial sentence by starting the generation from the verb's index to the end of the sentence. Store this information in a string called a partial phrase. Line 6 of the code performed exactly partial text generation and gives the output below.

output:

Partial phrase = views customer's profile.

The subsequent step followed was to eliminate the punctuation at the end of the sentence. This paper has harnessed the power of regular expression (regex) for text processing to eliminate the punctuation from the partial phrase. After that, replace the verb from the partial phrase with its verb lemma. The code below illustrates how to replace a verb with its lemma.

The ultimate step left is now to collect the pieces of information that are attributed towards the formation of the user story (subject and partial phrase]. Use this information to form a user story using the template in section 1.

From the template, we get the following user story when using sentence 1.

Template = [subject/actor] I want to be able to **[**partial phrase]

Subject: The bank Administrator

Partial phrase: view customer's profile.

It is also imperative for the user story formed to possess an object. This was extracted from the partial phrase's dependency graph by identifying words with the dependency keyword **obj**. If the object exists, use the user story's template to fill in the corresponding missing information to form the user story. The results of the user story for the above text were found as:

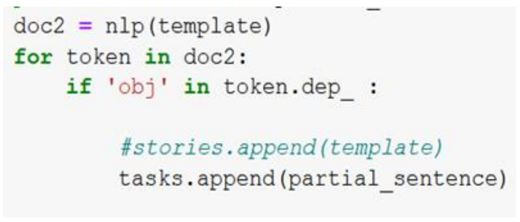
User story: As the bank Administrator I want to be able to view customers' profiles.

2) Algorithm for multiple verbs

In case the sentence is comprised of multiple verbs that are inactive voices, use a dependency graph to determine the subject or compound from the sentence given. Furthermore, use the dependency graph's metadata to extract the presence of root nodes that are verbs from the entire sentence. Save the index of each verb found from the input text. Then, initialise the splitting process to form partial phrases. The splitting process was summarised by the following steps below:

1. Get the root verbs and their index positions from the entire sentence. Save this information in a list
2. Count three words before reaching the next root verb in the sentence and append this text to the root verb.
3. Form partial sentences with the output of step 2.
4. Verify the presence of an object (obj) from the partial sentences by searching through the

dependency graph. We have illustrated this process by using the code snippet below.



```
doc2 = nlp(template)
for token in doc2:
    if 'obj' in token.dep_ :

        #stories.append(template)
        tasks.append(partial_sentence)
```

Fig. 6. Determine the presence of object in generated phrase

5. Refine all unnecessary conjunctions words that complicate the user story to avoid ambiguity.
6. From the partial sentence formed, replace the verb with its lemma.
7. With the information from step 6, retrieve the subject and fill in the user story's template with its corresponding text. Remember template = As [subject], I want to be able to [partial phrase].
8. Use chunking to extract the tasks from obtained user stories generated.
9. Repeat all the steps until the entire text is processed.

D. *Extracting tasks from the generated user stories*

After the generation of user stories is complete, we extracted useful insights from the decomposed user stories to form tasks associated with those decomposed stories. To decompose user stories into tasks, this paper builds on the guidelines provided by an empirical study on how to formulate tasks from a given user story by applying NLP techniques [16]. We have also distilled grammatical patterns that generate the task from the given user story by using the chunking technique through the aid of Spacy-stanza annotations and pipelines. To be more specific, verb phrase detection was the most effective chunking technique we employed. A verb phrase is a syntactic phrase that consists of at least one action verb. This verb can be trailed by other chunks, such as object phrases, noun phrases etc.

The tasks were extracted by analysing partial phrases from the user story information. This paper determined the rules that govern the determination of tasks from their respective user story linguistic structure linguistic task stricture.

1) The initial word should be verb a with dependency tag 'root'

Therefore, the pattern of finding the tasks from the user story was given as:

Pattern = 'r(<VERB>? <OBJ>*<NN>+)'

For demonstration purposes, consider the formulated user story in the section Algorithm for the single verb.

The output of the task was found as:

Task: view customer's profile.

IV. Experimental Evaluation

To evaluate the proposed algorithm, we provided the file which comprised of epic stories with different case studies. The first case study is a prominent cash withdrawal system called ATM (Automated Teller Machine). Secondly, we formulate another epic by using an eCommerce website. We adapted epic requirements specifications from the Agile Samurai textbook.

A. *Case study: ATM*

Input text: The bank customer can withdraw money from the ATM without card. The Bank customer can also deposit money on the ATM, change PIN on the ATM and transfer funds from the current account to the savings account on the ATM. The customer should be able to receive SMS notifications when money is withdrawn from the account.

1) *Generated stories and tasks*

TABLE I DECOMPOSED EPICS RESULTS FROM ATM TEXT

Index range	***Generated stories***	***Tasks***
4:11	As the bank customer I want to be able to withdraw money from ATM without card	Withdraw money from ATM without card
16:22	As the bank customer I want to be able to deposit money on the ATM	Deposit money on the ATM
22:27	As the bank customer I want to be able to change PIN on the ATM and	Change PIN on the ATM and
27:35	As the bank customer I want to be able to transfer funds from the current account to savings account on the ATM	Transfer funds from the current account to savings account on the ATM
35:42	As the bank customer I want to be able to use the ATM if money is	to use the ATM if money is
49: -1	As the bank customer I want to be able to receive SMS notification	Receive SMS notification

B. *Case study: Ecommerce*

Input: The customers should be able to view products sold online. If the customer decides to purchase the products online, add products to the bucket where s/he can continue with the purchasing process as the guest or create account for shipping purposes. They must be able to pay with visa cards or cash on delivery (COD). The system should validate expired cards to avoid scammers.

1) *Generated stories and tasks*

TABLE II DECOMPOSED EPICS RESULTS FROM ECOMMERCE

Index range	***Generated stories***	***Tasks***
6:8	As products I want to be able to view products.	View products
14:16	As products I want to be able to purchase the products online	Purchase products online
26:	As products I want to be able to add products to the bucket where s he can	Add products to the bucket where s he can
41:50	As products I want to be able to create account for shipping purposes they must be able to	create account for shipping purposes they must be able to

Index range	Generated stories	Tasks
65:69	As products I want to be able to validate expired cards to	validate expired cards to

V. RESULTS

This paper harnessed the power of the classical machine learning called confusion matrix metric in classification problems with known answers. To measure the performance, four metrics Accuracy, Precision, Recall, and F1 measure determined. Table III shows the overall performance of the tool with an aggregated average on the tested use cases.

TABLE III AVARAGE PERFORMENCE OF THE ALGORITHM

Metrics	*Percentage %*
Accuracy	89.25
Precision	100
Recall	77.25
F1 Measure	87%

VI. CONCLUSION

Based on the results obtained, this paper concludes that NLP is an adequate technique to automate Agile software artefact generation. We have obtained an average accuracy of 89.25%, an F1 measure of 87%. Through further experiments and analysis of sentence structure that formulates Agile epics, this paper infers that the linguistic structure of epics explicitly possesses two or more action verbs. However, this does not imply stories implicitly expressed. For example, " Administrator should be able to manage user account". The word manage can be extended into four different words such as delete, add, update, and create.

REFERENCES

[1] M. Kassab, “The changing landscape of requirements engineering practices over the past decade,” in *2015 IEEE fifth international workshop on empirical requirements engineering (EmpiRE)*, 2015, pp. 1–8.

[2] A. Azzazi, “A Framework using NLP to automatically convert User-Stories into Use Cases in Software Projects,” *Int. J. Comput. Sci. Netw. Secur.*, vol. 17, no. 5, pp. 71–76, 2017.

[3] R. P. Verma, “Generation of Test Cases from Software Requirements Using Natural Language Processing,” *6thh Int. Conf. Emerg. Trends Eng. Technolo*, no. 6, pp. 141–148, 2013, doi: 10.1109/ICETET.2013.45.

[4] C. Wang, F. Pastore, A. Goknil, and L. Briand, “Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach,” *IEEE Trans. Softw. Eng.*, pp. 1–1, 2020, doi: 10.1109/tse.2020.2998503.

[5] S. Dimitrijević, J. Jovanovic, and V. Devedžić, “A comparative study of software tools for user story management,” *Inf. Softw. Technol.*, vol. 57, no. 1, pp. 352–368, 2015, doi: 10.1016/j.infsof.2014.05.012.

[6] M. Ali, Z. Shaikh, and E. Ali, “Estimation of Project Size Using User Stories,” *Int. Conf. Recent Adv. Comput. Syst.*, no. Racs 2015, pp. 54–60, 2016, doi: 10.2991/racs-15.2016.9.

[7] M. Choetkiertikul, H. K. Dam, T. Tran, T. T. M. Pham, A. Ghose, and T. Menzies, “A deep learning model for estimating story points,” *IEEE Trans. Softw. Eng.*, 2018, doi: 10.1109/TSE.2018.2792473.

[8] K. Kuusinen, P. Gregory, H. Sharp, L. Barroca, K. Taylor, and L. Wood, *Agile Process in Software Engineering and Extreme Programming*, vol. 283. 2017. doi: 10.1007/978-3-319-57633-6.

[9] I. M. Ratner and J. Harvey, “Vertical slicing: Smaller is better,” in *Proceedings - 2011 Agile Conference, Agile 2011*, 2011, pp. 240–245. doi: 10.1109/AGILE.2011.46.

[10] L. Müter, T. Deoskar, M. Mathijssen, S. Brinkkemper, and F. Dalpiaz, “Refinement of User Stories into Backlog Items: Linguistic Structure and Action Verbs,” Springer, 2019, pp. 109–116. doi: 10.1007/978-3-030-15538-4_7.

[11] A. C. Pereira, “using NLP to generate user stories from software specification in natural language,” UNIVERSIDADE FEDERAL DO PARANÁ, 2018.