A FRAMEWORK TO IMPROVE USER STORY SETS THROUGH COLLABORATION

by

Salih Göktuğ Köse

B.S., Software Engineering, Bahçeşehir University, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2023

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my thesis advisor Assist. Prof. Fatma Başak Aydemir, for her patience, unlimited guidance, and continuous support of my research. This thesis would not be possible without her efforts.

Besides my advisor, my appreciation goes to my jury members: Prof. Tunga Güngör, and Assist Prof. Reyhan Aydoğan for their supportive comments and the engaging discussion during my defense.

I would like to express my gratitude to my parents Candan and Aytuğ Köse for being the best parents that a child might have. Their support made everything possible. I would like to thank Hilal Nur İşçen for her endless support, patience, and love. Without her unconditional support, this study would not be completed.

This work was partially supported by the Turkish Directorate of Strategy and Budget under the TAM Project 2007K12-873.

Salih Göktuğ Köse

ABSTRACT

A FRAMEWORK TO IMPROVE USER STORY SETS THROUGH COLLABORATION

Agile methodologies have become increasingly popular in recent years. Due to its inherent nature, agile methodologies involve stakeholders with a wide range of expertise and require interaction between them, relying on collaboration and customer involvement. Hence, agile methodologies encourage collaboration between all team members so that more efficient and effective processes are maintained. Generating requirements can be challenging, as it requires the participation of multiple stakeholders who describe various aspects of the project and possess a shared understanding of essential concepts. One simple method for capturing requirements using natural language is through user stories, which document the agreed-upon properties of a project. Stakeholders try to strive for completeness while generating user stories, but the final user story set may still be flawed. To address this issue, we propose SCOUT: Supporting Completeness of User Story Sets, which employs a natural language processing pipeline to extract key concepts from user stories and construct a knowledge graph by connecting related terms. The knowledge graph and different heuristics are then utilized to enhance the quality and completeness of the user story sets by generating suggestions for the stakeholders. We perform a user study to evaluate SCOUT and demonstrate its performance in constructing user stories. The quantitative and qualitative results indicate that SCOUT significantly enhance the quality and completeness of the user story sets. Our contribution is threefold. First, we develop heuristics to suggest new concepts to include in user stories by considering both the individuals' and other team members' contributions. Second, we implement an open-source collaborative tool to support writing user stories and ensuring their quality. Third, we share the experimental setup and materials to evaluate the SCOUT.

ÖZET

KULLANICI HİKAYE KÜMELERİNİ İŞBİRLİKÇİ ŞEKİLDE İYİLEŞTİREN BİR ÇERÇEVE

Çevik metodolojiler, son yıllarda giderek daha popüler hale gelmiştir. Doğası gereği çevik metodolojiler, geniş bir uzmanlık yelpazesine sahip paydaşları içerir ve aralarında işbirliğine ve müşteri katılımına dayanan etkileşim gerektirir. Bu nedenle çevik metodolojiler, daha verimli ve etkili süreçlerin sürdürülmesi için tüm ekip üyeleri arasındaki işbirliğini teşvik eder. Gereksinimlerin oluşturulması projenin çeşitli yönlerini tanımlayan ve önemli kavramlar hakkında ortak bir anlayışa sahip birden fazla paydaşın katılımını gerektirdiğinden işbirlikçi ortamlarda zor bir görev olabilir. Doğal dil ile gereksinimleri oluşturmanın basit bir yöntemi, bir projenin üzerinde uzlaşılan özelliklerini belgeleyen kullanıcı hikayeleridir. Paydaşlar, kullanıcı hikayeleri oluştururken bütünlük için çaba sarf ederler, ancak oluşturulan kullanıcı hikayesi seti yine de kusurlu olabilir. Bu sorunu ele almak için, doğal dil işleme metodları ile kullanıcı hikayelerinden anahtar kavramları çıkartan ve çıkarttığı bu kavramlar arasındaki ilişkileri kullanarak bir bilgi çizgesi oluşturan SCOUT'u öneriyoruz. SCOUT, oluşturulan bilgi çizgesi ve farklı buluşsal yöntemleri kullanarak paydaşlar için öneriler üreterek kullanıcı hikayesi setlerinin kalitesinin ve eksiksizliğinin arttırılmasını sağlar. SCOUT'u değerlendirmek ve kullanıcı hikayeleri oluşturma konusundaki performansını göstermek için bir kullanıcı çalışması gerçekleştirdik. Niceliksel ve niteliksel sonuçlar, SCOUT'un kullanıcı hikayesi setlerinin kalitesini ve eksiksizliğini önemli ölçüde artırdığını göstermektedir. Sağladığımız katkı üç yönlüdür. İlk olarak, hem bireylerin hem de diğer ekip üyelerinin katkılarını kullanarak kullanıcı hikayelerine dahil edilecek yeni kavramlar önermek için buluşsal yöntemler geliştirdik. İkinci olarak, kullanıcı hikayeleri yazmayı ve kalitelerini sağlamayı desteklemek için açık kaynaklı bir ortak çalışma aracı geliştirdik. Üçüncüsü, deney düzeneğini ve materyallerini paylaştık.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS/ABBREVIATIONS

BERT	Bidirectional Encoder Representations from Transformers
BFS	Breadth First Search
CrowdRE	Crowd-Based Requirements Engineering
CRUD	Create-Read-Update-Delete
DL	Deep Learning
ELMo	Embeddings from Language Models
GloVe	Global Vectors for Word Representation
GPT	Generative Pre-trained Transformer
HTTP	Hyper-Text Transfer Protocol
LTS	Long-Term Support
ML	Machine Learning
MLM	Mask Language Modeling
MVC	Model-View-Controller
NL	Natural Language
NLP	Natural Language Processing
NLP4RE	Natural Language Processing for Requirements Engineering
NoSQL	Non-SQL
NSP	Next Sentence Prediction
RE	Requirements Engineering
RQ	Research Question
SCOUT	Smart Collaborative User Story Tool
SOP	Sentence Order Prediction
SQL	Structured Query Language
STS	Semantic Textual Similarity
UC	User Interface Component
WSGI	Web Server Gateway Interface

1. INTRODUCTION

Requirements engineering practices aim to define the various aspects of a software project to ensure that it meets the necessary expectations. In the context of agile methodologies, multiple experts with diverse backgrounds contribute to the formation of viable requirements definitions for different aspects of the project. Collaborative tasks help stakeholders effectively utilize their time and resources [1]. However, aligning the independent activities of these stakeholders can be challenging, particularly when stakeholders are located in different places. While bringing stakeholders together physically may be a solution to this problem, it can be costly in terms of both time and money. This situation may be problematic for projects with strict deadlines and limited resources. Therefore, collaboration is critical in such environments.

Natural language is commonly used for representing requirements documents as it is less time-consuming and resource-intensive compared to other representations such as formal or semi-formal representations [2, 3]. User stories are a widely-used natural language notation for capturing requirements [3, 4]. Due to their simplicity of use and compliance with a predefined format, they are widely used by professionals in the industry [5]. During requirements engineering practices, a set of user stories is compiled. Practitioners attempt to ensure that this set covers every detail of the project. However, manually determining the completeness of the user story set can be challenging. In such cases, providing on-demand suggestions to stakeholders when creating user stories can save time and effort that would otherwise be spent examining the user story set. Therefore, an automated system that assists stakeholders in constructing a complete solution can be beneficial.

In collaborative environments, generating requirements is a complex process due to the need for multiple stakeholders to represent various aspects of the project, all of whom must possess a shared understanding of key concepts [6]. It is not feasible for stakeholders to continually evaluate the entire set of user stories while introducing

different aspects of a system. Additionally, ensuring that all stakeholders are on the same page and providing a coherent solution for a project can be challenging in such environments. To alleviate these difficulties for stakeholders, we propose the use of SCOUT to increase the completeness of user stories in collaborative environments. This system offers an interactive tool that generates on-demand suggestions by constructing a knowledge graph depending on the user input. The knowledge graph is created by employing a natural language processing (NLP) pipeline to extract key concepts from user stories and group them based on their relatedness. We then apply various heuristics to the information stored in knowledge graphs to generate suggestions for stakeholders. Similar to the approach of Aydemir and Dalpiaz [7], our pipeline relies on natural language text so that incorporating additional features in the future will not require extensive changes. Our pipeline measures the similarity of different noun phrases using sentence embeddings to match slightly different terms that refer to the same concept. We extract keywords using a deep language model and use various algorithms to connect related terms and then construct the knowledge graph. By leveraging the knowledge graph and various heuristics, we generate suggestions for stakeholders to enhance the overall completeness of user stories for a project.

Our contributions with this thesis are as follows:

(i) A web interface with a simplistic design that enables stakeholders to construct requirements artifacts easily.
(ii) An NLP pipeline that processes the raw requirements texts. This pipeline extracts concepts from user stories.
(iii) A graph module that constructs knowledge graphs via defining the relationships between these concepts using the state-of-the-art BERT deep language model and storing these relations between concepts.
(iv) A suggestion module employs various heuristics to generate suggestions for enhancing the completeness of the user story set, leveraging the knowledge graphs.
(v) A publicly available user story set. [8]
(vi) A publicly available source code. [9]

Preliminary work for my thesis [10] that focuses on glossary extraction from collaborative requirements models to support collaboration and help stakeholders while aligning multiple models, is published in the Proceedings of the 11^{th} International Workshop on Model-Driven Requirements Engineering (MoDRE). My unpublished work for this thesis is also shared in arxiv [11] to fulfill the requirements of the degree of master of science. I plan to submit this work to an international peer-reviewed journal.

The remainder of the thesis is structured as follows. Chapter 2 provides background information for methods used in this study. Chapter 3 reviews the related work. Chapter 4 explains our research approach. Chapters 5 - 10 focuses on the details of SCOUT. Chapter 11 reports on the evaluation plan. Finally, Chapter 12 concludes the thesis.

2. BACKGROUND

This chapter provides background information on the methods used in this thesis. Section 2.1 focuses on natural language processing along with pre-processing steps in natural language processing and state-of-the-art natural language processing techniques. Graph databases are explained in Section 2.2.

2.1. Natural Language Processing

The area of computer science known as natural language processing (NLP) focuses on applying computational methods to learn, understand, and create material in human language [12]. Some well-known applications of NLP consist of machine translation, text summarization, language generation, chatbots, and text classification. Since NLP depends on human language, several challenges arise. Khurana *et al.* [13] report that one of the major challenges in NLP is the variability and complexity of human language, which makes it difficult for computers to accurately process natural language. However, recent developments in deep learning have significantly enhanced the performance of NLP systems [14].

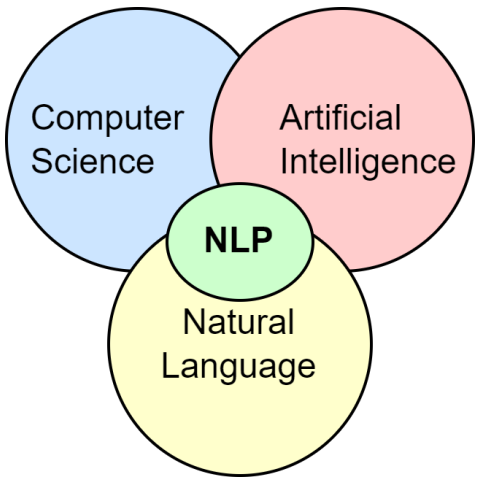

Figure 2.1. NLP in a nutshell.

2.1.1. Pre-processing in NLP

Pre-processing is an important step in any NLP task, as it helps to clean, normalize and prepare the raw text data for further analysis. Pre-processing steps may include:

- Case-folding: Changing all words to lowercase might be helpful since it minimizes the dimensionality of the data and reduces the possibility that two equal words would be treated differently because of their capitalization.
- Stop word and punctuation removal: Stop words are highly common yet redundant words such as "the" and "of" [15]. To make the data smaller and increase the performance of the NLP algorithms, they might be eliminated from the text. However, Yu [16] argues that stop words can serve as distinguishing characteristics for specific tasks. According to Etaiwi and Naymat [17], punctuation marks are used frequently in text and they have an influence on the results of text processing strategy, particularly those that depend on the occurrence of words and phrases.
- Stemming and lemmatization: Jivani [18] notes that lemmatization and stemming both strive to reduce a word to its base form and also states that lemmatization considers part of speech and context, but stemming does not. Both of these techniques can be useful for reducing the dimensionality of the data and improving the performance of NLP algorithms.

Overall, pre-processing steps are crucial since these steps help to clean and prepare the raw data for further steps of the implementation. We carefully consider which pre-processing steps are suitable for our task, as they can significantly impact the quality of the output.

2.1.2. State-of-the-art NLP techniques

Otter *et al.* [19] point out that machine learning techniques were frequently employed to conduct NLP tasks in the past yet during the past several years, there has been a complete shift, and neural models have replaced or at least improved traditional techniques. According to Xipeng *et al.* [20], recent studies have shown that pre-trained models on a large corpus may acquire universal language representations without the need to build a new model from the beginning. The early usage of pre-trained models depends on word embeddings that are context-free such as GloVe [21] and ELMo [22]. Recently, contextual word embeddings such as GPT [23] and BERT [24] are widely used. Figure 2.2 shows the pre-training steps of BERT.

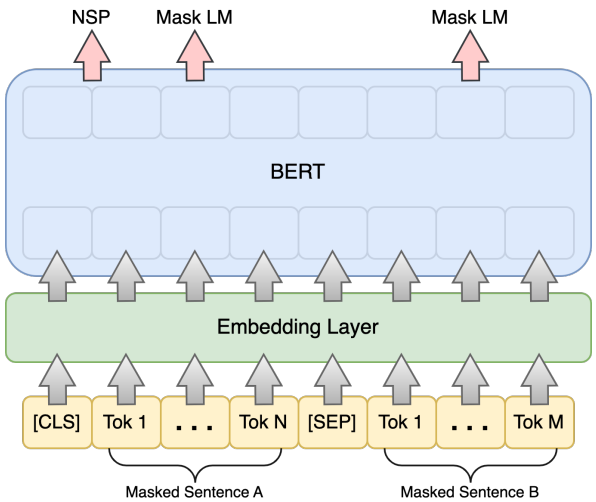

Figure 2.2. Pre-training steps BERT based on [24].

Transformer architectures make it easier to create high-capacity models which depend on training bidirectional transformer models on huge unlabeled text corpora, and pre-training has allowed researchers to utilize this capacity efficiently for a range of applications [25,26]. This approach underlies models like BERT [24], RoBERTa [27], DistilBERT [28] and ALBERT [29]. Müller *et al.* [30] state that, mask language modeling (MLM), next sentence prediction (NSP), and sentence order prediction (SOP) are generally used to perform unsupervised training to obtain a general language model that can be utilized to perform particular language processing tasks such as classification, question-answering models, and chatbots.

2.2. Graph Databases

NoSQL databases that employ graph structures of nodes, edges, and attributes to represent and store data are known as graph databases. Vukotic *et al.* [31] state that graph databases are utilized in a range of applications, such as recommendation engines, fraud detection, and social network analysis, and are especially helpful for processing complex and connected data. Figure 2.3 demonstrates a simple graph. Blue nodes represent actors and yellow nodes represent movies. The relationships between movies and actors are represented with *ACTED_IN* relation.

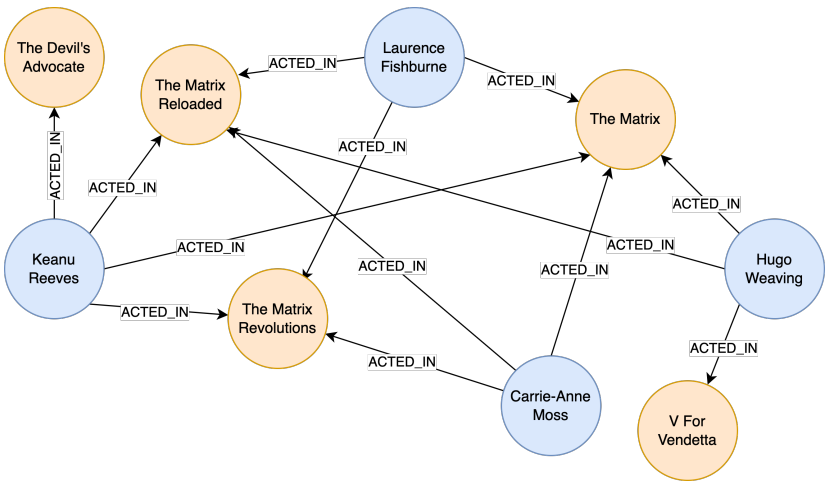

Figure 2.3. Sample graph representation.

The potential of graph databases to exhibit complex relationships in data is one of its main advantages. When dealing with huge and connected data sets, foreign keys, which are normally used in a standard relational database to describe relationships, may be challenging to handle and extremely slow to query [32]. Contrarily, graph databases represent relationships using nodes and edges, which makes it simple to store and query complex data structures. This makes graph databases ideal for real-time recommendation engines and other applications that process massive amounts of data in real-time. We choose neo4j, a popular open-source graph database, to store connected data since graph databases are powerful for storing and analyzing complex relationships in data. Additionally, a comparison between different graph database models made by Angles [33], reveals that neo4j performs more efficiently compared to other implementations.

3. RELATED WORK

This section presents the selected works from four main related areas: agile requirements engineering, collaboration in software engineering, NLP for requirements engineering, CrowdRE, and gamification.

3.0.1. Agile Requirements Engineering

Kassab *et al.* [2] state that a vast number of practitioners prefer natural language for software requirements representation. Among the free, semi-structured, and structured formats, user stories, which follow a semi-structured notation, are widely preferred by developers adopting agile methodologies [3, 4] due to their simplicity. The popularity of the user stories is the main motivation behind our research.

Despite their simple structure, user stories are not always high quality. Lucassen *et al.* [34] state that user stories are frequently poor in quality. As with any artifact constructed with natural language, user stories suffer from ambiguity.

Stakeholders often spare less time to construct requirements than needed. Therefore, defects like ambiguity and incompleteness emerge commonly in software requirements. Dalpiaz *et al.* [35] propose the REVV-light tool that presents terms in requirements in different viewpoints using Venn diagrams to mainly detect ambiguities and partially detect incompleteness in user stories. Similarly, we focus on a common requirements defect as incompleteness in our research. Additionally, we employ a deep language model to generate suggestions for stakeholders to pinpoint the cause of incompleteness in user stories. Yang *et al.* [36] propose a method to detect anaphoric ambiguity caused by different interpretations of pronouns by different stakeholders via implementing different NLP heuristics. Barbosa *et al.* [37] focus on detecting duplicate user stories that are present in an agile development cycle via using information retrieval metrics along with semantic similarity measures. Yang *et al.* [38] propose a

tool that employs a machine learning algorithm to detect coordination ambiguity in requirements texts. Literature shows that defects in natural language requirements are widely studied areas. However, a vast number of studies mainly focus on the intrinsic defects of user stories. Our method focuses on increasing the completeness of user story sets.

In large-scale software projects, stakeholders often require a better understanding of the requirements definitions. To support the identification of requirements artifacts, conceptual models and goal-oriented approaches are commonly used. Trkman *et al.* [39] experiment with human experts to investigate the effectiveness of conceptual models -specifically business process models- compared to textual requirements and point out that participants achieve equal or better results when a BPMN model is provided. Güneş and Aydemir [40], propose a method for capturing relations among user stories via automatically generating goal models using an NLP pipeline to eliminate the human effort required for constructing models. Wautelet *et al.* [41] state that due to its simple structure, a large number of user stories are needed in large-scale projects. This leads to several difficulties such as inconsistency. They propose a method that generates a rationale diagram based on i* framework components that allow easier user story analysis. Jaqueira *et al.* [42] state that user stories have a limited capability for representing and detailing requirements and they propose a method to create a broader viewpoint for stakeholders via enriching user stories with i* framework. Wautelet *et al.* [43] experiment with different target groups to investigate the difficulties that modelers encountered while generating goal-oriented models from user story sets. Throughout their experiment, the authors opt for completeness as an evaluation criterion.

Literature states that user stories are widely used in agile development processes. Even though user stories follow a simple structure, agile practitioners need support when dealing with user stories in large-scale environments to prevent defects such as incompleteness and ambiguity. The basis of our research depends on supporting stakeholders when constructing user stories in terms of completeness.

3.0.2. Collaboration in Software Engineering

Teruel *et al.* [44] define collaborative systems as systems that allow users to work cooperatively while performing tasks. Whitehead [45] states that stakeholders engage in a range of activities throughout the software development process including the creation of software artifacts such as source code, models, documentation, and test scenarios. Therefore, coordinating numerous stakeholders' efforts is essential to form a viable environment as well as producing more quality software products. However, collaboration is challenging in such environments consisting of stakeholders with different backgrounds and even different cultures in distributed software projects.

Constantino *et al.* [46] claim that distributed software projects suffer from several issues in terms of collaboration and also point out that despite these challenges stakeholders in collaborative environments produce better artifacts compared to any single developer's effort. Herbsleb [47] also states that distributed development environments are negatively affected in terms of coordination and discusses the activities designed to provide coordination among stakeholders. Collaboration in distributed software development environments is provided by various tools to automate and facilitate the entire development process. Lanubile *et al.* [48] state that collaboration tools let stakeholders work together even when stakeholders work apart from each other and provided a list of collaborative development tool categories as (i) version-control systems, (ii) trackers, (iii) build tools, (iv) modelers, (v) knowledge centers, (vi) communication tools and (vii) Web 2.0 applications.

Konaté *et al.* [49] state that requirements engineering procedures are interdisciplinary processes that necessitate specialized skills from all stakeholders and draw attention to the importance of collaboration in requirements engineering since a single stakeholder cannot satisfy all of the skills required for the job. Azadegan *et al.* [50] state that stakeholders in teams need to work collaboratively and also point out the fact that each individual has a different background and perspective. Even though maintaining a collaborative environment is difficult, collaborative methods in requirements

engineering produce better results. With the emergence of crowd-based requirements engineering (CrowdRE), collaboration becomes more crucial since the essence of CrowdRE relies on employing larger numbers of individuals to increase user involvement. Hosseini *et al.* [51] state that coordination becomes a problem when requirements engineers have to engage with a big group of users and argue that a major challenge is developing software-based mechanisms to coordinate the crowd with minimal developer participation while being cost-effective.

Literature shows that collaboration is a key concept for software engineering as well as requirements engineering. However, maintaining a collaborative environment is challenging due to the diversity that can be observed among agile teams. We aim to reduce the difficulties of collaboration by providing suggestions to stakeholders depending on their effort as well as their effort as a team so that SCOUT mitigates the risks that might arise due to defective interaction between stakeholders.

3.0.3. NLP for Requirements Engineering

Applying requirements engineering techniques in a software project is crucial since a great portion of the projects throughout history either have failed or exceeded the budget due to the lack of proper requirements definitions [1]. Davis *et al.* [52] prove that software requirements definitions affect the quality of the product significantly. Lamsweerde [53] points out the increasing importance of requirements engineering in software engineering since defining requirement properly is a challenging task.

Dalpiaz *et al.* [54] state that natural language (NL) is widely used in requirements engineering (RE) due to the ease of understanding even by inexperienced stakeholders besides the ambiguity of NL. It can be inferred that NLP complements RE so that a detailed analysis might be conducted. The close relationship between the fields of NLP and RE leads to the field of NLP4RE emerging. Zhao *et al.* [55] define NLP4RE as "an area of research and development that seeks to apply NLP techniques requirements artifacts". There are numerous studies in the literature in the area of NLP4RE. Ferrari

and Esuli [56] propose a method for detecting cross-domain ambiguities in RE by building domain-specific language models to approximate the potential ambiguities. Duan *et al.* [57] propose a method for automated requirements prioritization technique based on clustering requirements by their probability of distribution and co-occurrence. Robeer *et al.* [58] propose the Visual Narrator tool that generates conceptual models from user stories by combining several NLP heuristics. Hotomski and Glinz [59] propose the GuideGen approach that generates natural language guidance when the acceptance test needs to be aligned with the requirements definitions. Gemkow *et al.* [6] propose a method for glossary term extraction from large-scale requirements documents by using several NLP techniques and statistical filtering.

Recent studies show interest in applying deep learning and machine learning methods in the field of RE due to the low generalization of traditional methods. Stanik *et al.* [60] propose a comparative approach between supervised machine learning (ML) and deep learning (DL) methods by applying these methods to a user feedback classification task and reported that supervised ML performed slightly better than the deep learning method due to the limited number of training samples. By their nature, deep learning methods require a larger number of training samples compared to supervised ML models. However, pre-trained models seem to provide a solution to this problem. Hey *et al.* [61] propose a fine-tuned version of the BERT deep language model as NoRBERT for requirements classification and reported highly promising results in classification tasks in the non-functional requirements dataset. Araújo *et al.* [62] also propose a RE-specific version to extract requirements from app reviews and reported that their fine-tuned version of the BERT language model outperforms all of the existing methods.

Surveyed literature shows that NLP4RE is an emerging and widely studied area. Furthermore, pre-trained models convey a high potential for several tasks in requirements engineering. We aim to contribute NLP4RE area by applying an NLP pipeline that employs a pre-trained deep language model.

3.0.4. CrowdRE and Gamification

User involvement has favorable consequences on system success and it is shown that employing users as an information source is an effective way to elicit requirements [63, 64]. According to El Emam *et al.* [65], more user participation reduces the detrimental impact of ambiguity on the quality of requirements engineering tasks. Kujala *et al.* [64] point out that increasing user involvement in requirements elicitation activities yields well-formed requirements so that the chance of project success is increased. Crowdsourcing is a scalable and inexpensive solution that increases user involvement in RE processes. [66, 67]

However, providing healthy interaction between users is challenging. To increase user interaction towards systems, gamification is widely applied in many areas such as finance, education, and health [68]. Gamification is also applied in the area of requirements engineering. Snijders *et al.* [67] state that gamification complements crowdsourcing by motivating users to participate more by rewarding the users. Fernandes *et al.* [69] propose the gamified environment iThink, to support collaborative requirements elicitation. Snijders *et al.* [66] propose REfine online platform that is enriched with gamification techniques to facilitate requirements elicitation tasks.

Surveyed literature shows that involving greater numbers of users increases the number and quality of requirements. However, the collaboration between users and user interaction with the system becomes more difficult with the increasing number of users. Our method provides an easy-to-use user interface and several functionalities such as highlighting user stories with detected problems to increase user interaction with the system. In future work, we plan to employ additional gamified objects in SCOUT.

4. RESEARCH APPROACH

Although user stories are widely adopted in the industry [3, 4, 70] and studied in academia [5,34,35,37], existing work focus on the user stories as a set by either checking the quality [34, 71, 72] or extracting models from the final set [39–43]. We identify a gap in the literature regarding collaboratively building a user story set rather than analyzing a complete set of user stories. This discovery leads us to our main research question: "How can we support stakeholders in a collaborative environment while constructing user story sets?".

To address our main research question, we design our study following Wieringa's three-step cycle of the design science research approach [73] as summarized in Figure 4.1.

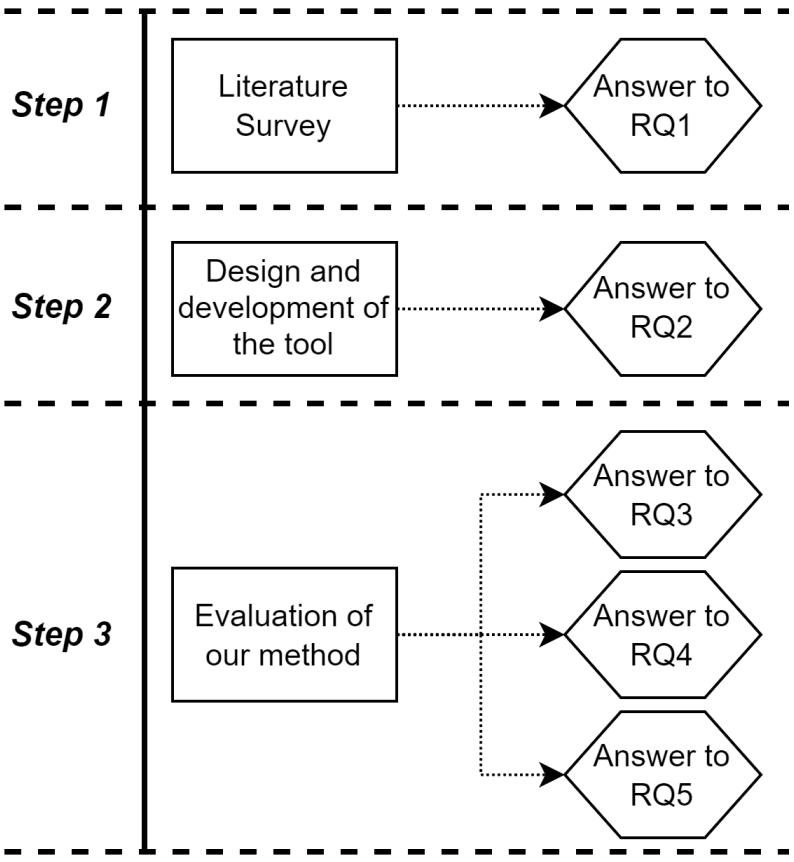

Figure 4.1. Steps of our research approach.

In this study, we address the following research questions:

- RQ1: What are the issues of the current agile requirements engineering practices?
- RQ2: What is an effective artifact that can support stakeholders when constructing user stories?
- RQ3: How effective is our system in terms of ensuring the completeness of the user story set?
- RQ4: To what extent do the participants implement the suggestions produced by SCOUT?
- RQ5: What is the perceived usability level?

To answer RQ1, we conduct *problem investigation* task by performing a thorough literature survey. We represent the issues with the status quo in Table 4.1

Table 4.1. Issues of the current agile requirements engineering practices.

Issue ID	Description
Issue 1	Difficulties arise from natural language
Issue 2	Effort required for fixing defects
Issue 3	Need for better tools for collaboration

Surveyed literature clearly states:

(i) Even though the widespread usage of NL requirements representations, practitioners often tend to construct error-prone solutions even with simple structured notations such as user stories due to the ambiguity that arises from the nature of natural language. [2–4, 34, 70] To control the quality of user story sets, several methods are proposed but they require practitioners to be trained. [34, 71, 72, 74, 75]

(ii) Several methods that employ NLP techniques are proposed to detect defects in requirements documents. [35–38] To reduce the time spent fixing defects in requirements documents, a method that helps practitioners to improve user stories as they are written, is needed.

(iii) Software development processes and requirements engineering practices require stakeholders to work cooperatively on different tasks to produce better output [44–48]. Requirements engineering practices are interdisciplinary processes and the emergence of CrowdRE requires a large number of practitioners to work collaboratively [49–51]. However, the increasing sizes of agile teams along with members in separate locations make preserving viable collaborative environments harder. Software-based smart tools can be developed to provide a potential solution to this issue.

We aim to improve the status quo concerning the second step of Wieringa's design cycle, *design a treatment*. To answer RQ2, we propose SCOUT. SCOUT is a collaborative requirements editor that supports stakeholders by generating on-demand suggestions. Our method is detailed in the rest of the paper. We decide to create a tool that has a simple UI that allows users to interact with the system easily. On top of this, we instrument SCOUT with a state-of-the-art NLP pipeline that employs a pre-trained deep language model. Finally, we perform *treatment validation* for the design artifact as suggested by Wieringa's design cycle.

The efficient use of resources in large-scale agile teams greatly depends on the level of collaboration among the stakeholders yet maintaining collaboration is a challenging task in distributed environments [1, 46, 47]. Collaboration needs to be supported with different instruments. When it comes to requirements engineering, practitioners with diverse skills put effort to satisfy the goals of the project. [49, 50] An increased level of collaboration reflects positively on the quality of the requirements artifacts [63, 64]. However, the collaborative environment needs to be properly maintained otherwise several defects such as incompleteness might occur in a user story set [34].

RQ3 aims to measure the effectiveness of SCOUT in terms of completeness. We benefit from the collaborative input of the participants to generate our suggestions to increase the completeness of the user story sets. We analyze the number of user stories and standard deviation of user stories along with the number of isolated concepts that

are present among user stories. An increased number of user stories and a reduced number of isolated concepts have a positive impact on the completeness of the user story set. Also, we treat the graphs generated by our graph generation module as trees and we applied breadth-first search (BFS) traversal. The increasing number of nodes in BFS traversal indicates that users provide more detailed properties for each concept that is supported with different sentences.

RQ4 aims to measure the portion of the suggestions that are accepted as useful and applied by the participants. This enables us to measure the effectiveness of our suggestion strategies. SCOUT aims to increase the completeness of the user story sets by providing suggestions to stakeholders. SCOUT implements different heuristics to generate suggestions for users. Thusly, we assess the success of the heuristics as well as SCOUT itself, by measuring the level of incorporation of the generated suggestions. We generate different suggestions for the participants throughout the experiment sessions. We expect users to apply some of these suggestions that contribute to their way of thinking. The adoption ratio of the suggestions is directly proportional to the effectiveness of SCOUT.

RQ5 aims to understand whether users consider SCOUT beneficial. We directly address an industry problem hence SCOUT must be useful for requirements engineering professionals. To answer RQ5, the participants are asked several questions on a 5-point Likert scale in the post-experiment survey to evaluate the effectiveness of SCOUT, the user interface, and the level of usefulness of provided suggestions in terms of quality and completeness.

5. SCOUT: SUPPORTING COMPLETENESS OF USER STORY SETS

Our goal in this work is to create a web-based collaborative agile requirements tool that supports stakeholders as they create user stories in a shared environment. We develop SCOUT to achieve this goal. Stakeholders can create and store user stories with SCOUT so they can use them to gather suggestions and make the user story set more complete. Stakeholders may be assigned to teams and projects so they can carry out various tasks for various projects. Both the individual efforts of stakeholders and the whole set of user stories for a project that is created by all stakeholders are benefited by SCOUT. To create an environment that is viable for stakeholders, we implement various modules and submodules. Web services are used to provide a communication channel between the editor and the modules. Figure 5.1 demonstrates the architecture of SCOUT.

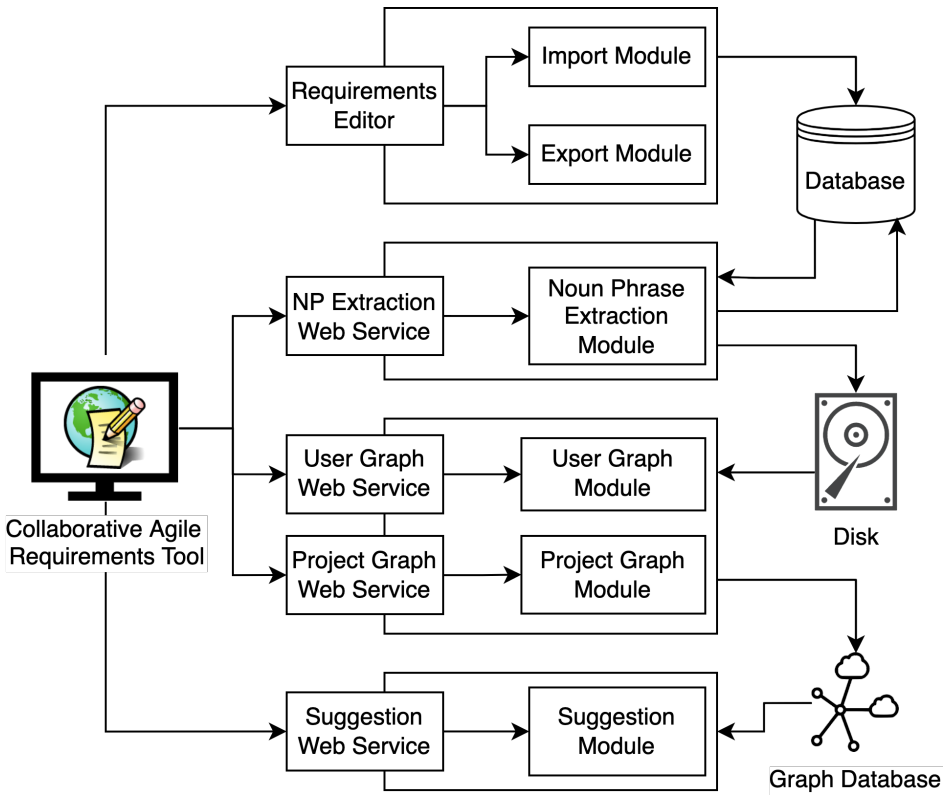

Figure 5.1. Architecture of the SCOUT.

Different modules of the SCOUT are explained as follows:

- Collaborative Agile Requirements Editor: We implement a collaborative agile requirements editor for stakeholders. We instrument our interface with two different sub-modules as import and export modules. These modules serve to increase the usability of the system. Stakeholders can easily import user story sets in hand and export them into different file formats. The user interface also allows stakeholders to gather suggestions with a single click, and chat with other stakeholders.
- Noun Phrase Extraction Module: We implement a noun phrase extraction module as an initial step to extract the most informative parts of user stories. Our noun phrase extraction module is responsible for processing raw natural language text in user stories into noun phrases and transferring the extracted information to the database.
- Graph Generation Module: We aim to transform the output of the noun phrase extraction module into the concepts present in the user story set using the relatedness between the extracted noun phrases. To achieve this goal, we implement an NLP pipeline that depends on pre-trained BERT sentence embeddings. After we extract concepts from noun phrases, we store these concepts and their relations in a graph database. We use this information in the graph database for visualization and further steps of our implementation. We implement two different web services to populate the graph database for each stakeholder and project.
- Suggestion Module: We aim to generate suggestions to increase the completeness of the user story set. To achieve this goal, we benefit from the relations between concepts that are stored as knowledge graphs. We implement different heuristics to generate suggestions for stakeholders using individual stakeholders' input along with all stakeholders' input for a project.

The internal workings of SCOUT's noun phrase extraction, graph, and suggestion modules are shown in Figure 7.1, Figure 8.1, and Figure 9.1, respectively.

6. COLLABORATIVE AGILE REQUIREMENTS EDITOR

We implement an interactive interface that increases user interaction with the system. Figure 6.1 presents a screenshot of the interface of our system.

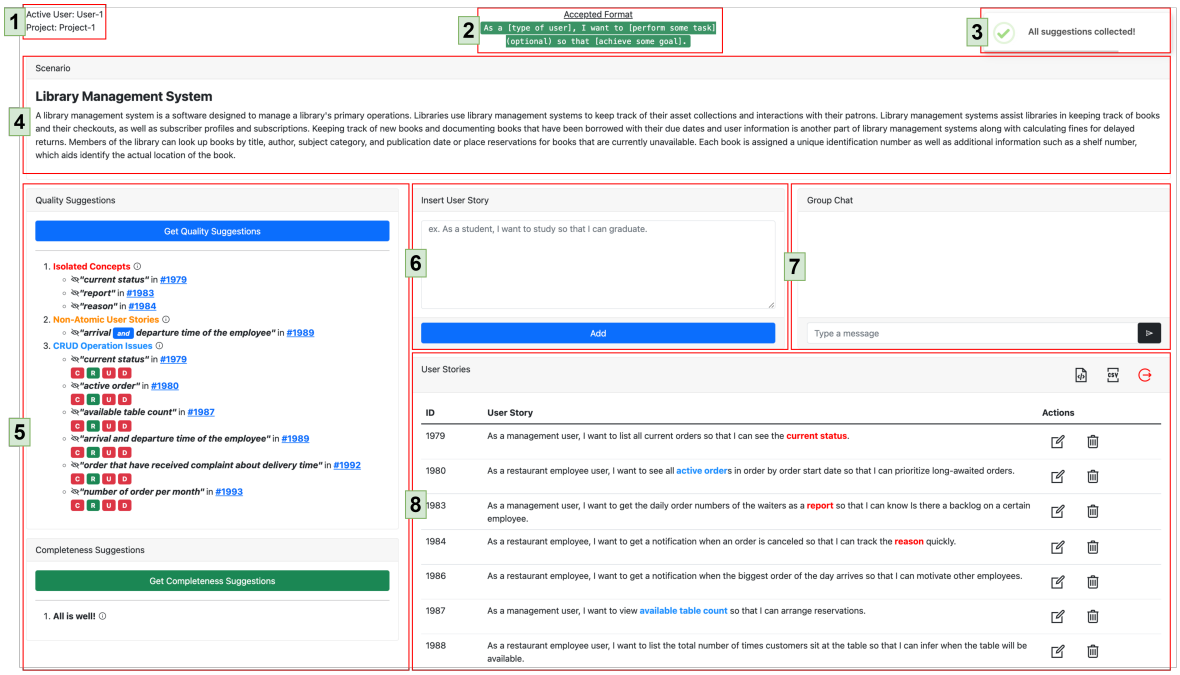

Figure 6.1. User interface of the collaborative requirements editor of SCOUT.

We explain the different user interface components (UC) as follows:

- UC #1: This component shows the information about the user that is logged in and which project the user is currently working on.
- UC #2: This component shows the accepted user story format since errors might occur due to format violation. We add this component on top of the main page which helps users quickly control whether the sentence complies with the format. We also check the format with a regular expression. If the user does not enter a valid user story, the input user story will be available in UC #5 for editing.
- UC #3: This component shows different messages for different actions that can be performed by the user. It informs users whether the action they perform results in success or failure.

- UC #4: This component holds the scenario text associated with the project. We put the scenario text in the middle of the screen to enable users to check that text without any disturbance.
- UC #5: This component is used to request and display suggestions. Users can request individual and group suggestions from SCOUT. These suggestions are listed in different areas in this component. SCOUT provides two types of individual suggestions listed as isolated and atomic; eight types of group suggestions listed as CRUD, close-to-completeness, pop-zero, pop-one, pop-two, pop-three, feeling lucky and all is well to the users. Detailed information about suggestion types is provided in Chapter 9. Suggestions are displayed in this panel and divided into separate categories. Users can report suggestions that are not useful for their purpose by clicking on the crossed-eye icons. Users also can easily navigate the sentences related to an individual suggestion by clicking on the hyperlinks at the end of each individual suggestion.
- UC #6: This component is used to add new user stories to the system. If the added user story does not comply with the user story format, this component will be automatically filled with the sentence again to allow the user to quickly fix that sentence.
- UC #7: This component is used to chat with other members of the same team. Sent messages are displayed in real-time for all users of the same team to provide a viable communication channel.
- UC #8: This component is mainly used to display user stories that are constructed by the logged-in user. Users can edit and delete user stories. Users can also export these user stories as JSON and CSV formats using the buttons in the upper right corner. Figure 6.1 shows that some of the user stories have colorful elements. To identify the elements in sentences that may need to be fixed, these parts are generated based on the suggestions that are obtained.

7. NOUN PHRASE EXTRACTION MODULE

The noun phrase extraction module analyzes the natural language text in user stories. Since stakeholders create user stories using unconstrained NL, a list of noun phrases is extracted from the raw NL text via our NLP pipeline. Our noun phrase extraction pipeline is summarized in Figure 7.1 and the steps of our noun phrase extraction module are detailed in the rest of this section.

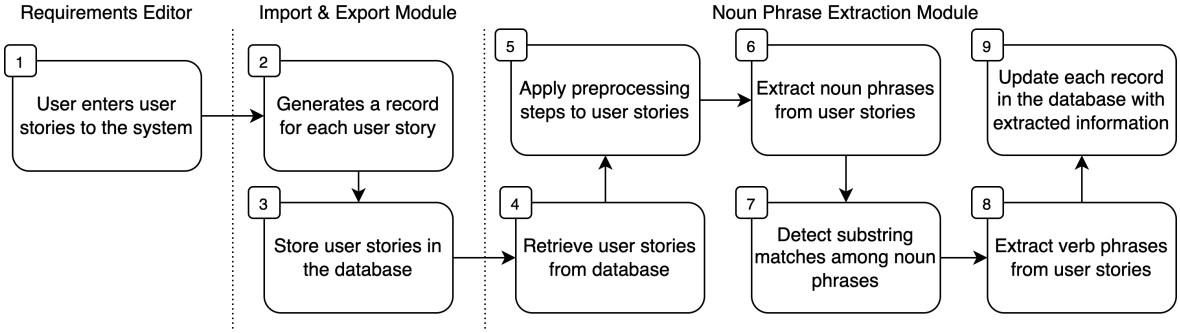

Figure 7.1. Pipeline of inner workings of noun phrase extraction module.

Our pipeline starts with user interaction. (Step 1) Users can add single or multiple user stories to the system from the user interface. (Step 2) With the help of the import module, we parse the user input and create records that involve user, project, and time information for each record. After that, we store these records in a database. We also provide an export module for users to export their input as JSON or CSV formats. (Step 3) Our noun phrase extraction pipeline starts by retrieving these user stories by querying the database. (Step 4)

For step 5, we use several pre-processing techniques to enhance the extraction of noun phrases from user stories. Our noun phrase extraction module includes several steps: (i) case-folding to normalize the user stories. (ii) punctuation removal to eliminate unnecessary information, (iii) lemmatization to reduce inflectional forms of words. We choose lemmatization due to its dependence on both context and morphological aspects of words, unlike stemming. Stop-word removal is widely used in NLP tasks [55]. Nevertheless, following several test runs, we decide against removing stop words from

the user stories. Even though stop-word removal generally leads to better results for NLP tasks, in our case doing so produces fairly poor results because the absence of conjunctions and prepositions results in parts of sentences' merging unintentionally. This produces meaningless noun phrases that affect our other steps of implementation. We put these steps into implementation by utilizing NLTK and native Python libraries to discover the concepts that exist in user stories and to create a knowledge graph.

For step 6, we extract noun phrases from user stories. Our initial method for capturing the information in the user stories is to extract nouns as concepts from the pre-processed user stories. However, because of the inadequate information provided by nouns alone, working exclusively with nouns does not produce satisfactory results. Therefore, we choose to use the spaCy [76] to extract noun phrases from user stories. For each user story, we extract dependency trees and noun chunks. We combine each extracted noun chunk with its syntactic descendants to broaden the scope of our noun phrase extraction module. To accomplish this, we concatenate all of the syntactic descendants of each extracted noun chunk from the leftmost edge to the rightmost edge of the dependency tree.

For step 7, we detect substring matches among noun phrases. We discovered that there are many noun phrases with parts in common after extracting noun phrases from user stories. They are what we refer to as substring matches that are observable between noun phrases. These noun phrases with parts in common can be grouped using the subset of noun phrases that contain a specific noun phrase as a substring. This process can be simply described as noun phrase clustering. For instance, the noun phrases "item label" and "item label size" can be grouped under the "item label" key since "item label size" refers to a property of "item label". By using this step, all noun phrases are grouped into a set, with the shortest noun phrase serving as their representation. The number of noun phrases in the initial set is decreased by applying this step and this reduction also reduces the time and space complexity of our further steps of implementation.

For step 8, we extract verb phrases from user stories. We discover that verb phrases in user stories also contain important information after extracting noun phrases. We develop a sub-module in our pipeline for extracting noun phrases that extract this data from verb phrases. Our preliminary tests, however, show that not all verb phrases provide information that is pertinent to our goal. We develop the concept of verb phrase extraction for CRUD (create-read-update-delete) operations that can be used on an artifact after conducting a thorough investigation. The completeness of the user story set could be jeopardized by the absence of coherent CRUD operations. For instance, unless it has been created beforehand, no project artifact can be deleted. As shown in the example, this technique satisfies our initial goal of making the user story set more complete. However, this operation cannot be performed with just four verbs, so we must develop a glossary of terms that can be used in place of the four primary operations. Our glossary is created using a thesaurus found online. After that, we extract verbs from user stories using the pattern-matching technique of textacy [77]. Then, to extract complete verb phrases from user stories, we navigate the syntactic descendant tree. After retrieving verb phrases, we divide them into four main categories referred to as CRUD.

For step 9, we update records in the database with the extracted information. After applying all of the steps of our noun phrase extraction pipeline. Our noun phrase extraction module updates each record with the extracted information from the user stories.

8. USER AND PROJECT GRAPH MODULES

We implement a graph module that uses an NLP pipeline powered with a deep language model to extract concepts from the output of the noun phrase extraction module and store this data in a graph database. To process the work of individual stakeholders as well as the entire project user story set, which is built by numerous stakeholders, we develop two distinct web services. Our graph module uses the extracted concepts and their relations to populate the graph database. Our graph module is summarized in Figure 8.1 and the steps of our graph module are detailed in the rest of this section.

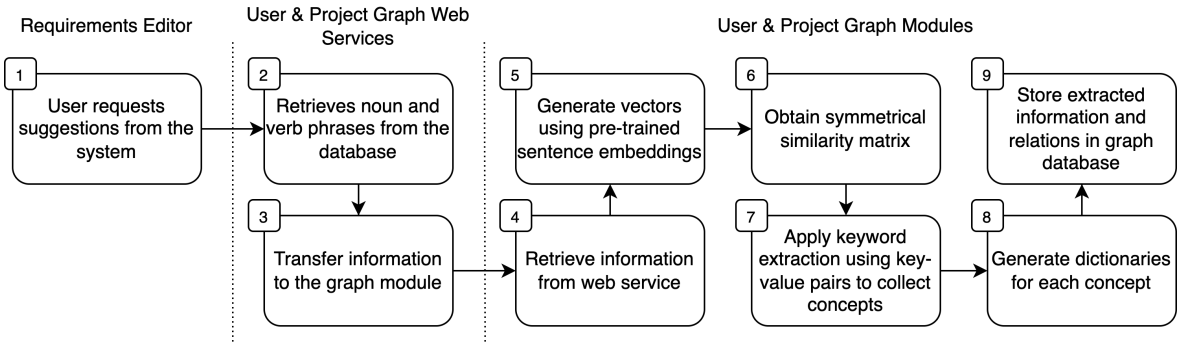

Figure 8.1. Pipeline of inner workings of graph modules.

Our pipeline starts with user interaction. Users can request suggestions from the interface of SCOUT. (Step 1) This request is processed by our user & project graph web services which act as our middleware layers. These services retrieve concepts to the graph module. (Step 2) The extracted information is transferred to the user & project graph modules. (Step 3) Our graph module starts by retrieving the information from the web service. (Step 4)

For steps 5 and 6, we create sentence embeddings. In the noun phrase extraction module, we are able to group a few of the user stories although independent user stories still exist. Although requirements engineering experts created the user stories, experts with diverse backgrounds may use different words to describe the same idea. For instance, due to their close resemblance, the noun phrases "shipping offers" and

"shipment options" will be matched as similar noun phrases. We take advantage of BERT, a state-of-the-art deep language model [24], to reveal this relation and broaden the scope of concept extraction. As a result, knowledge graph extraction from user stories is made possible by comparing the similarity of various noun phrases to one another and by identifying various noun phrases that refer to the same meaning using pre-trained sentence embeddings.

First, Hugging Face's [78] sentence transformers, which are intended to be used with a trained model, are used to encode noun phrases. For sentence embedding, many pre-trained sentence embedding models are available. We select *paraphrase-mpnet-base-v2* [79] because of its great performance on the STSbenchmark test. For each noun phrase, this sentence encoding process produces vectors with 768 dimensions. Pairwise cosine similarities between each noun phrase are calculated to extract similar noun phrases, and the symmetrical similarity matrix is then obtained.

The goal is to collect the noun phrase pairs with the highest similarity after obtaining the similarity matrix. To achieve this, we create an algorithm that iteratively creates pairs of noun phrases that are the most similar to one another. Figure 8.2 presents the algorithm for sentence embeddings. The number of candidate similar noun phrases is limited with a pre-defined similarity threshold to reduce the number of iterations necessary. The predefined similarity threshold is set to 0.4 after several test runs. (Line 2) The similarity score between a noun phrase and each candidate noun phrase is taken from the similarity matrix during each iteration and compared with a noun phrase that is previously chosen and the target noun phrase. (Lines 7 - 9) Only in cases where the new noun phrase's similarity score exceeds both the threshold and the previous similarity score will the target noun phrase and similarity score be updated. (Lines 10 - 12)

There may be a smaller common unit of representation for the noun phrases in the key-value pairs, even though the majority of similar key-value pairs are extracted from the noun phrases. With a pre-trained sentence embedding model, KeyBERT [80] is used

```
Input  : a dictionary L of noun phrases and a matrix S of pairwise
         similarities
Output: a dictionary D of similar noun phrase pairs
D = {};
threshold = 0.4;
foreach term T in keys of L do
    similarity_score = -1;
    P = '';
    terms = list of terms in the similarity matrix S
    foreach term N in terms do
        if T is not equal to N then
            score = similarity score between T and N
            if score > similarity_score and score > threshold then
                similarity_score = score;
                P = N;
    end
    P = keywordExtraction(P, T);
    if P is not empty then
        append T to D[P]
    else
        append P to D[T]
    end
end
append dictionary L into dictionary D
foreach key K in keys of D do
    if D[K] is empty then
        delete D[K]
end
return D;
```

Figure 8.2. Algorithm for sentence embeddings.

to extract keywords from the key-value pairs to provide a higher level of abstraction. We again select *paraphrase-mpnet-base-v2* [79] because of its great results on both our sentence embedding step of the graph module and the STSbenchmark test. (Line 14)

```
   Input  : parent term P and term T
   Output: parent term
1  parent = P;
2  term_string = T + parent;
3  keywords = extracted keyword and probability pairs from term_string using
     KEYBERT
4  if length of keywords > 1 then
5  |   set the most probable keyword as parent
6  return parent;
```

Figure 8.3. Algorithm for keyword extraction.

For step 7, we apply keyword extraction to collect concepts. To extract the most representative n-grams from the texts, keyword extraction techniques are used. Figure 8.3 presents the algorithm for keyword extraction. With this algorithm, lemmatized versions of each pair's key and value are gathered and joined together to create a single string (Line 2). KeyBERT is adjusted to extract unigrams from that concatenated string (Line 3). Both the key and the value are grouped under the new representation if there is a smaller common unit of representation for a key-value pair. (Lines 4 - 5)

For step 8, we generate dictionaries for each concept. Related and parent terms are kept in a dictionary after using keyword extraction. The parent terms will be the dictionary's keys, and the terms that are related to the parent terms will be their values. Each term is added to the dictionary as a value of the parent term. (Lines 15 - 16) When a term lacks a related term with a similarity score greater than the threshold, we append it as a parent term, and its value is set as a list with an empty string. They will be handled as separate terms with no relation to other terms. (Lines 17 - 18). The output of the noun phrase extraction module is added to the dictionary

that is created during the iteration. (Line 21) To get rid of terms that the noun phrase extraction module did not process, we delete key-value pairs with empty values. (Lines 22 - 24)

For step 9, we store the extracted information in a graph database. A graph database is used to store all concepts and the relationships between them that are extracted from the user stories. Since identifying the relationships between concepts is our main goal, we decide to store the extracted concepts in a graph database because doing so enables us to gain insightful information. To store our data, we choose the open-source Neo4j graph database, and to access the database from our graph module, we employ the Neo4j Bolt Driver [81]. We build two distinct databases for user and project graphs so that we can examine the user story set from various angles. For both databases, we define the same node structure that has the following attributes: (i) key as the concept; (ii) user_id as the user's unique identifier; (iii) project_id as the project's unique identifier; (iv) user_story as the list of user stories related to the concept; (v) is_active as the flag indicates whether that node is generated by the latest commit; (vi) expiry_date as the timestamp denotes the time that record becomes inactive. is_active and expiry_date properties are aligned with each other. Consider a scenario in that a user requests suggestions for the first time. In that case nodes will be created with $is\_active = 1$ and $expiry\_date = 9999\text{-}12\text{-}31$ properties. If the user request suggestions again in that state, before adding new nodes existing nodes with $is\_active = 1$ property are converted to $is\_active = 0$ and their $expiry\_date$ is set to the timestamp of that time. Implementing this logic enables us to keep track of the changes in graphs over time. To identify related concepts as edges between nodes, we define a $RELATED\_TO$ relationship that denotes the edges between nodes.

A sample representation of the extracted concepts is shown in Fig 8.4. Red vertices (self-links) indicate isolated concepts, while blue vertices (matching concepts) indicate this.

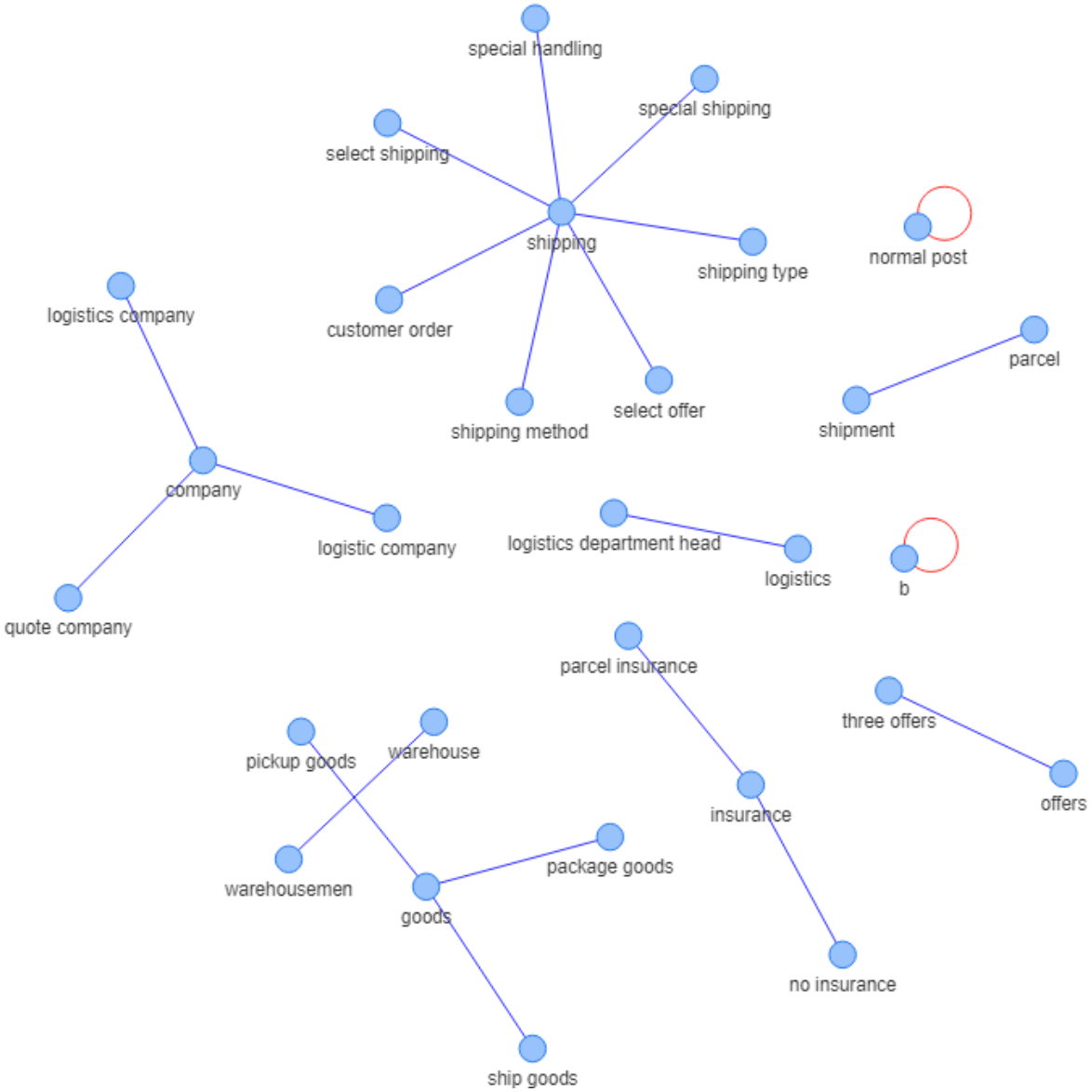

Figure 8.4. Sample graph representation of extracted terms.

9. SUGGESTION MODULES

We implement a suggestion module to generate suggestions by applying several heuristics to the information that is stored in the graph database. We provide ten different types of suggestions divided into categories as individual and group suggestions to the stakeholders by querying and traversing the graph database. Our suggestion module is summarized in Figure 9.1 and the steps of our suggestion module are detailed in the rest of this section.

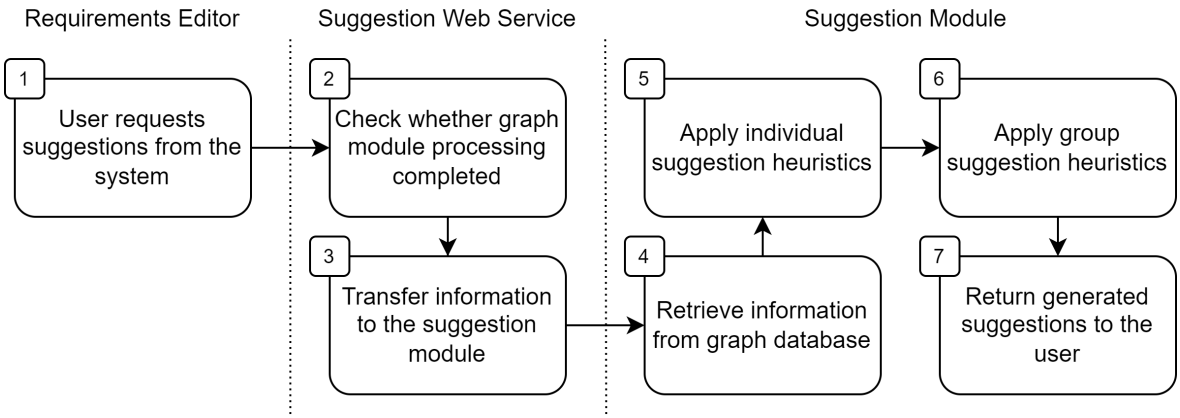

Figure 9.1. Pipeline of inner workings of suggestion module.

Our pipeline starts with user interaction. (Step 1) After the user requests suggestions from the system. The suggestion web service waits for the graph module to be completed since the suggestion module depends on the output of the graph module. (Step 2) After the graph module execution is completed, the suggestion web service transfers the information to the suggestion module. (Step 3) Consequently, the suggestion module queries the graph database to gather the required information. (Step 4)

9.1. Step 5 - Individual Suggestions

We aim to assist stakeholders to construct more quality user stories with our individual suggestion heuristics. We focus on the user story set of each stakeholder separately when generating individual suggestions.

Our first heuristic is the *isolated concepts* heuristic. We extract the list of concepts from user stories via noun phrase extraction and graph modules. We observe that some of the concepts do not have a relationship with any other concept. We refer to them as isolated concepts (self-links) and denote them with red edges in our graph representation. In other words, we collect the nodes that only have a relationship with themselves. Unlike other terms, these concepts have the potential for being unrelated when the scope of all user stories that are created by a single stakeholder is taken into account.

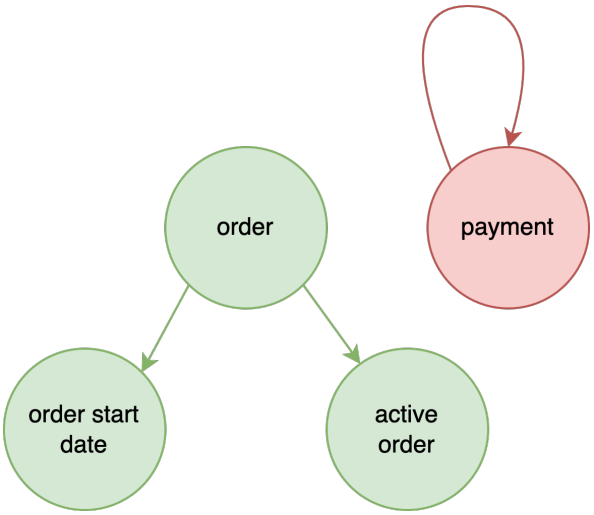

Figure 9.2. Representation of Isolated concepts heuristic.

Our initial aim with this heuristic is to get rid of unrelated terms so that stakeholders will be able to construct unambiguous user stories. To achieve this goal, we label these terms as isolated concepts and suggest stakeholders add more user stories about them or remove user stories about these concepts. Figure 9.2 represents the isolated concepts heuristic.

Our second heuristic is the *non-atomic user stories* heuristic. User stories are widely used for describing requirements simply and understandably. Thus, it is commonly accepted that user stories are moderately smaller compared to raw requirements texts. However, stakeholders sometimes tend to use more nouns, adjectives, prepositions, and conjunctions than necessary in user stories. As stated by Lucassen *et al.* [34], a user story should specify exactly one feature. Thus, we develop a heuristic that cap-

tures noun phrases that contain conjunctions so which eliminates the possibility of breaking the atomicity. By doing so, we manage to avoid misconceptions among stakeholders. We implement this heuristic by capturing the noun phrases that contain the conjunctions and & or. Figure 9.3 represents the non-atomic user stories heuristic.

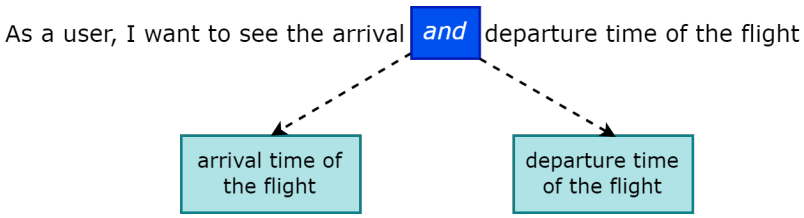

Figure 9.3. Representation of Non-atomic user stories heuristic.

9.2. Step 6 - Group Suggestions

We aim to increase the completeness of the user story set by assisting stakeholders with suggestions. To achieve this goal, we collect all user stories that are constructed for a project by different stakeholders and compare the whole user story set with each stakeholder's user story set to ensure that all stakeholders complement each other to achieve a better user story set with minimal effort.

Our third heuristic is the *CRUD operation issues* heuristic. By their nature, user stories define the actions that can be performed by actors of the software product. In some cases, user stories complement each other when the whole functionalities are considered. It can be inferred that the co-existence of some user stories that define the same part of the software system is a must. For instance, when a stakeholder creates the user story "As a user, I want to view my profile so that I can keep track of my progress.", it can be said that this profile must be created beforehand so that a user can view that profile. Therefore, the action of viewing a profile is dependent on the creation of that profile.

We define four main operations as CRUD (create-read-update-delete) and construct a glossary that contains synonym verbs for each operation. We extract verb phrases from user stories to look for verbs that influence the same term. If that term

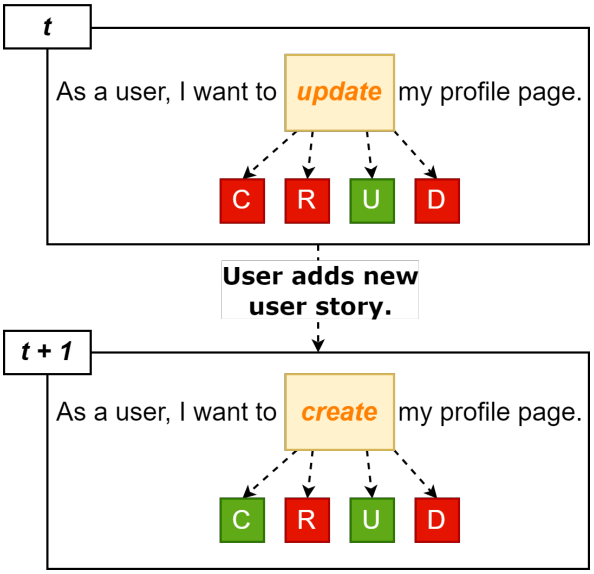

Figure 9.4. Representation of CRUD heuristic.

lacks complete CRUD operations, we generate a suggestion for stakeholders to define missing operations for that term. Figure 9.4 represents the CRUD heuristic.

We also benefit from nodes and edges in the graph database that are previously populated via our noun phrase extraction and graph modules. We implement different heuristics that benefit from graph traversal. We extract the most common (top-N) concepts from the project graph as a starting point. We treat them as our main concepts since a vast majority of stakeholders opt for these concepts while constructing user stories. To limit the number of suggestions that will be generated, we limit the number of extracted main concepts to 5. With the main concepts for a project at hand, we extract related concepts from the knowledge graphs.

We extract nodes that hold RELATED_TO relation with our main concepts. After several runs, we decide to deepen our graph traversal by employing Dijkstra's shortest path algorithm toward our undirected graphs stored in the graph database. We treat each of the main concepts as a root node and construct separate graphs for these main concepts by traversing through the child nodes until reaching the pre-defined maximum depth which is limited to 2 to limit the size of retrieved nodes. We use extracted graphs to construct a dictionary with main concepts as keys and child nodes

as values. After obtaining the dictionaries for user and project graphs. We initially compare keys in the dictionaries to compare the main concepts for a user story set.

When the user and project graph holds the same main concepts,

Our fourth heuristic is the *close-to-completeness* heuristic. It can be inferred that the user stories that are created by a stakeholder are in alignment with the whole user story set that is constructed by all stakeholders of a project. When the main concepts are the same, it is beneficial to compare the concepts that are related to the main concepts. We compare child node values as related concepts and identify missing concepts. We suggest users consider adding these missing child nodes of the main concepts.

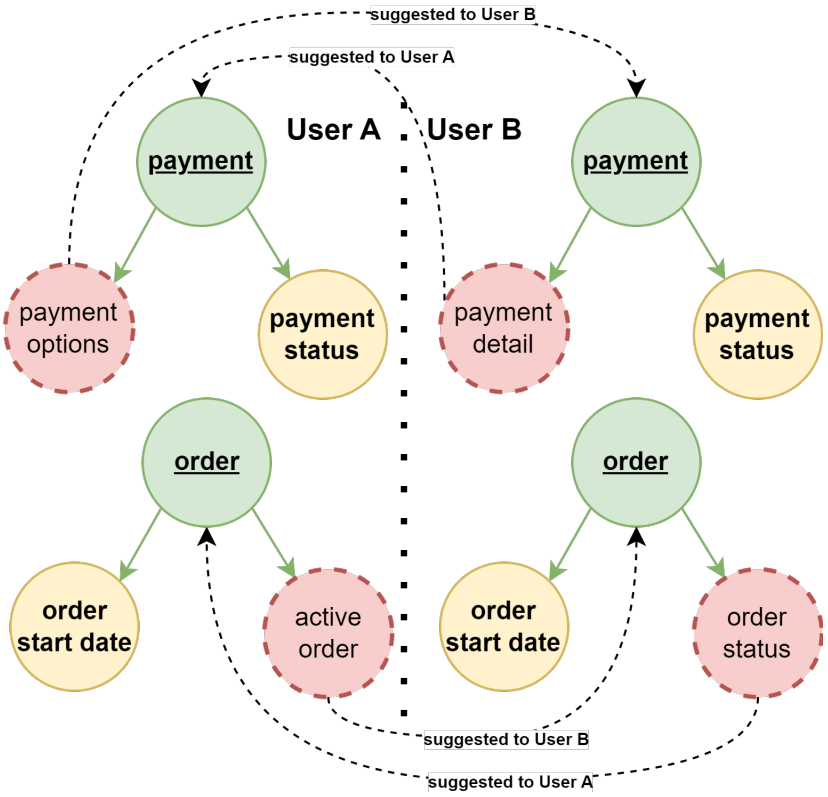

Figure 9.5. Representation of Close-to-completeness heuristic.

When the user and project graph do not hold the same related concepts for any of the main concepts, it can be inferred that a stakeholder mentioned different concepts than other stakeholders. Introducing stakeholders with concepts that they do

not include in their user story sets helps them to easily construct new user stories so that their input enriches the whole user story set.

Figure 9.5 represents the close-to-completeness heuristic. In Figure 9.5, User A and User B employ the same main concepts; "payment" and "order". However, they decide to construct user stories that denote different properties of these main concepts. Hence, the "close-to-completeness" heuristic benefits from the difference in related concepts to offer new ideas to users. Unlike User B, User A does not mention the concept of "order status". SCOUT generates a suggestion for User A using this heuristic to inform that another stakeholder defines the "order status" concept related to the main concept of "order". This heuristic allows stakeholders to gain a different perspective when needed.

Our fifth heuristic is the *pop-zero* heuristic. We consider the main concepts that are extracted from individual stakeholders' input along with their collective input. The difference in the main concepts leads the user story set to form an incomplete solution and also indicates that stakeholders do not share a common way of thinking.

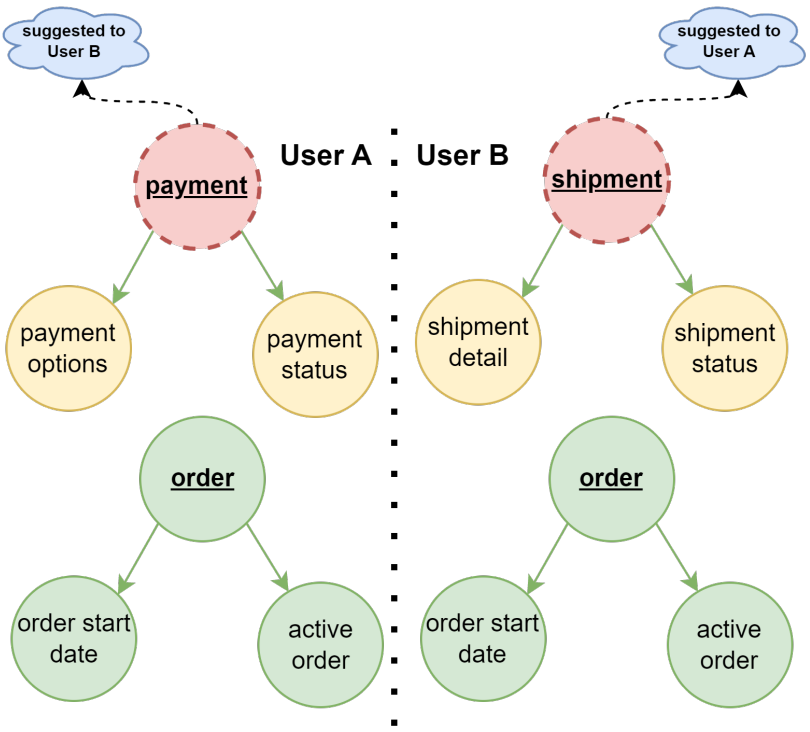

Figure 9.6. Representation of Heuristic 5.

Heuristic 5

Popular concepts that are less mentioned by the user.

In Figure 9.6, Figure 9.7, and Figure 9.8, User A and User B employ different main concepts; "payment" and "order" concepts are defined by User A, and "shipment" and "order" concepts are defined by User B. Since we consider the most popular five concepts as the main concepts of the whole user story set, the main concepts of the whole user story set consist of "payment", "shipment", and "order" concepts (These figures present only three for it is a toy model). On the other hand, we consider the main concepts of the user story sets generated by stakeholders separately, it is evident that these user story sets represent different main concepts.

Mitigating this issue helps stakeholders to enhance the completeness of the user story set. Therefore, we implement this heuristic that concerns the difference in the main concepts. This heuristic suggests stakeholders introduce new user stories concerning the main concepts that need to be emphasized in detail. Figure 9.6 represents the pop-zero heuristic.

In this state, the pop-zero heuristic benefits from this difference in main concepts. SCOUT generates a suggestion to inform a stakeholder that other stakeholders construct user stories that mainly focus on other concepts so that difference in main concepts is notified. The main concept of "shipment" is only utilized by User B thus SCOUT generates a suggestion for User A using the pop-zero heuristic to inform that other stakeholders constructed user stories that mainly focus on the concept of "shipment". This allows stakeholders to be introduced to concepts that are not covered by them thus providing insights by using other stakeholders' ideas.

Our sixth heuristic is the *pop-one* heuristic. In addition to the main concepts, we also take concepts that are related to the main concepts into account. In other words, we benefit from the main concepts and their related concepts. To uncover new concepts

contributed by other stakeholders, we extract differences in these related concepts.

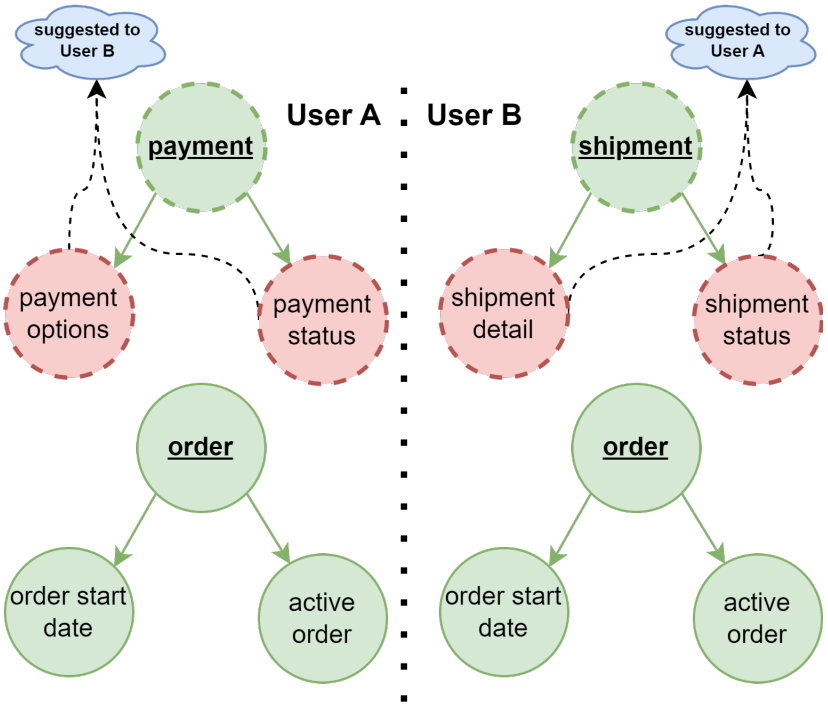

Figure 9.7. Representation of Heuristic 6.

Heuristic 6

Concepts that are not mentioned by the user but are related to the most popular concepts of the project.

This heuristic aims to assist stakeholders by exposing new ideas that are not utilized. By considering these suggestions, stakeholders can be aware of different aspects of a concept that is proposed by their teammates. Figure 9.7 represents the pop-one heuristic.

In this state, the pop-one heuristic benefits from this difference in main concepts to reveal the concepts that are related to these different main concepts. SCOUT generates a suggestion that provides concepts that are related to these different main concepts to complement the pop-zero heuristic. The main concept of "shipment" is only utilized by User B and User B constructs user stories that concern "shipment detail" and "shipment status" concepts. SCOUT generates a suggestion for User A using

the pop-one heuristic to provide concepts that are related to the main concepts of the project yet apart from the input of User A. This allows stakeholders to be introduced to related concepts not being limited to the main concepts and clearly understand what other stakeholders cover so which helps them to complement each other.

Our seventh heuristic is the *pop-two* heuristic. Besides the main concepts, we consider concepts that have a relation with the main concepts. We extract the main concepts and their related concepts. To uncover new concepts contributed by other stakeholders, we extract differences in these related concepts.

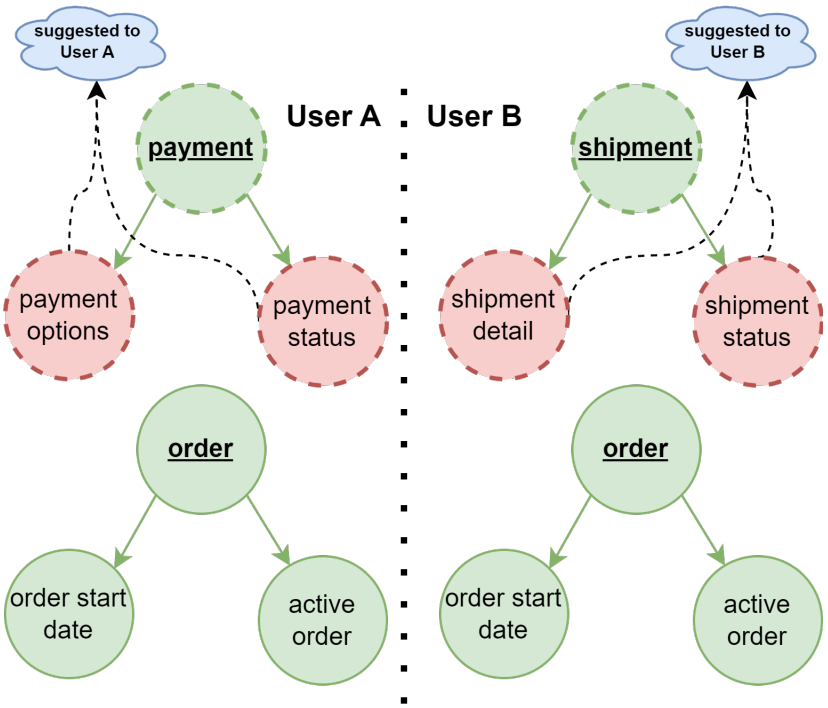

Figure 9.8. Representation of Heuristic 7.

Heuristic 7

Concepts that are only mentioned by the user and are related to the most popular concepts of the project.

This heuristic aims to assist stakeholders in two aspects by detecting concepts that are only utilized by them. Since any other stakeholder utilizes them in their user story sets, (i) more user stories about these concepts are necessary for better

clarification, (ii) these concepts can be discarded since they might be out of the scope of that project. Figure 9.8 represents the pop-two heuristic.

In this state, the pop-two heuristic benefits from this difference in the main concepts to reveal the concepts that are related to these different main concepts and utilized by that stakeholder alone. SCOUT generates a suggestion that provides related concepts to these different main concepts to the stakeholder which are only utilized by him/her. The main concept of "payment" is only utilized by User A and User A constructs user stories that concern "payment options" and "payment status" concepts. SCOUT generates a suggestion for User A using the pop-two heuristic to inform that these concepts are only utilized by himself/herself and might be unrelated to the main focus of the project. This might not always be the case yet we implement this heuristic as a precaution. Stakeholders are able to acknowledge that some of the user stories might be out of the project scope and needs to be removed or have better justification.

Our eighth heuristic is the *pop-three* heuristic. When the main concepts of the project are not the same as the main concepts of the stakeholder's input, concepts for some of the stakeholder's input cannot be evaluated. Hence, some of the concepts might be disregarded by the suggestion module.

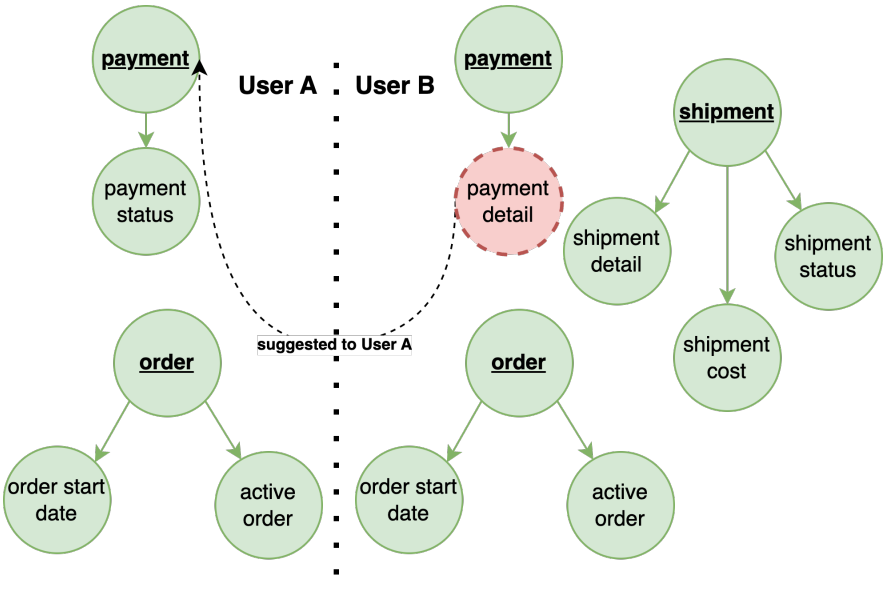

Figure 9.9. Representation of Heuristic 8.

Heuristic 8

Concepts that are related to the user's most used concepts are constructed by other members of the project.

To prevent some of the concepts from being disregarded, we compare the stakeholder's main concepts with the other stakeholders' main concepts, besides comparing the whole user story set's main concepts. By doing so, we enable stakeholders to gather suggestions even if a concept is not widely utilized by all of the members. The idea behind this is to help stakeholders to widen their views and introduce new concepts and user stories. Figure 9.9 represents the pop-three heuristic.

To illustrate this heuristic, we assume that we collect the most popular two concepts as the main concepts for a user story set for the sake of simplicity. In Figure 9.9, "payment" and "order" concepts are defined by User A, and "payment", "shipment" and "order" concepts are defined by User B. Stakeholders construct user stories that define the same part of the system but the concept "payment" is less mentioned by User B. Even though "payment" is not a main concept for User B, related concepts to "payment"—"payment details" in Figure 9.9— is suggested to User A for "payment" is indeed a main concept for this user. Gathering these concepts to suggest related concepts to users can serve to increase the completeness of the whole user story set.

Our ninth heuristic is the *feeling lucky* heuristic. When we generate individual suggestions, we extract isolated concepts from knowledge graphs. We generate suggestions about these isolated concepts that are created by other stakeholders for the particular user. This strategy generates the opportunity of collecting other users' opinions about isolated concepts and giving hints to the stakeholders by introducing extremely unfamiliar concepts that might turn into a better perception of the project scope. When the isolated concepts are not explained in detail by stakeholders, the whole user story set might not provide a complete solution. Figure 9.10 represents the feeling lucky heuristic.

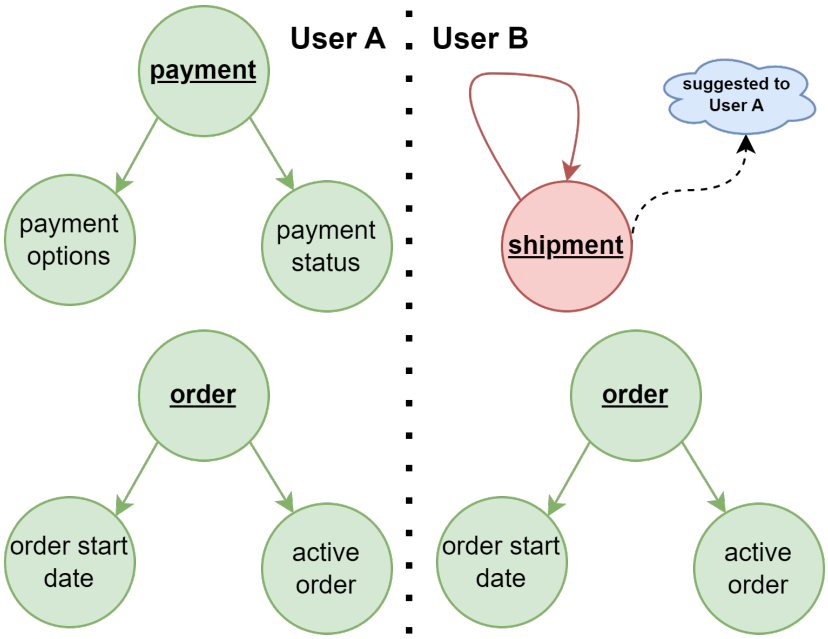

Figure 9.10. Representation of Heuristic 9.

SCOUT also fosters creativity by suggesting concepts to be included that are rarely mentioned by users. In Figure 9.10 "shipment" is an isolated concept, only mentioned by User B, and is not related to any other concept. Assuming the user might have overlooked this concept, SCOUT suggests it to User A. Seeing this concept may spark creativity for User A and may include it in the user story set.

Our tenth heuristic is the *all is well* heuristic. This heuristic is responsible for informing stakeholders that they have successfully achieved a complete solution. We consider both the main and related concepts of the individual stakeholder and the collective input of all stakeholders. When there is not any difference between the main and related concepts at the same time, it can be inferred that the individual stakeholder is aligned with the complete user story set.

For step 7, we return the generated suggestions to the users. After the suggestion module generates suggestions using the extracted information. It transfers suggestions to the user interface so that suggestions are displayed to the user.

10. IMPLEMENTATION

We use a Linux server that has 1 TB of memory along with an Intel Xeon Gold 6238 CPU with 22 cores that operates at 2.1 GHz. This server runs on Ubuntu 18.04 LTS as the operating system. This server allocates the resources depending on the fair shares principle.

We implement SCOUT using state-of-the-art technologies. Requirements editor implementation is held using ASP .NET Core. We choose .NET Core for being a lightweight, open-source, and platform-independent framework. We choose commonly used MVC as an architectural pattern. We implement the chat functionality of SCOUT using ASP .NET Core SignalR which is an open-source library for real-time message broadcasting. We employ a Microsoft SQL Server as the database for SCOUT. By their nature, .NET Core and Microsoft SQL Server work in harmony. To ease data management, we benefit from Entity Framework Core along with Code-First Approach. Therefore, creating the database from scratch takes only seconds, and managing operations without writing complex SQL queries provides a flexible implementation.

We implement web services using Python and Flask. We choose Flask for being a lightweight web framework that can be easily deployed on WSGI HTTP Servers. We choose gunicorn which is a speedy HTTP Server for deployment. We employ neo4j as a graph database for being open-source and providing high performance. We use neo4j Bolt Driver and Cypher Query Language to extract information from the graph databases.

We containerize the different parts of SCOUT using Docker to increase maintainability and management. We use official Docker images for Microsoft SQL Server and neo4j. We manually create Docker containers from scratch by constructing different Dockerfiles for different modules of the system. As a result, we use 4 different docker containers that Docker Compose orchestrates.

11. EVALUATION

This section presents the design, execution, and results of the conducted experiment. Our goal is to create an effective method compared to a standard collaborative text editor in terms of constructing user stories.

We hypothesize that H_0 : "The level of completeness for a user story set constructed with SCOUT is lower or equal to a user story set constructed with an average collaborative text editor". The one-tail alternative hypothesis is H_1 : "The level of completeness for a user story set constructed with SCOUT is greater than a user story set constructed with an average collaborative text editor".

Hence, we provide an efficient way of constructing a complete user story set for requirements engineers. Therefore, we evaluate our system by conducting experiments with human participants to measure the impact of SCOUT on the task of constructing user story sets.

11.1. Experimental Design

To evaluate SCOUT, we designed an experiment that involves participant engagement with the system. Participants are randomly assigned to different setups among two available setups. The experiment requires participants to write as many user stories as possible within a period. Figure 11.1 and Figure 11.2 presents the experimental design of SCOUT.

11.2. Procedure

We plan to conduct online experiment sessions with the participants via an online conference tool due to Covid-19 restrictions. We inform potential participants about our approach via a written document that contains a brief explanation about SCOUT

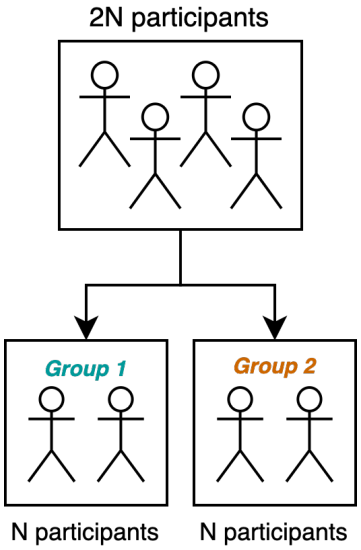

Figure 11.1. Participant distribution.

2N experiment sessions

Control Group	1st Execution	Google Docs	Scenario #1
	2nd Execution	SCOUT	Scenario #2
Treatment Group	1st Execution	SCOUT	Scenario #1
	2nd Execution	Google Docs	Scenario #2

Scenario #1 : Restaurant Management System
Scenario #2 : Library Management System

Figure 11.2. Experimental design.

along with the experimental setup. We also inform the potential participants about the average duration of the experiment so that they can arrange their available periods, if possible. After that, we invite them to participate in our experiment voluntarily. We try to gather participants who mostly have similar backgrounds. However, we also conduct experiments with participants that work outside of the IT industry and students to increase the coverage of our experiment. We aim to widen the application area of SCOUT as well as encourage other parties to employ user stories to support their business activities. We arrange available time slots with those who volunteered to be a participant in our experiment. Before the experiment, we get each participant's consent to participate in the experiment. We also collect demographic information from participants to recognize our target group.

After completing the pre-experimental steps, the experimenter makes a presentation that includes the aim of the experiment, experimental setup, experimentation timeline, and usage of the SCOUT along with the logic behind the different suggestion types. Participants are asked to complete the experiment on their own. We provide chat functionality in SCOUT to provide a communication channel between participants. Additionally, they do not require any additional tool or application to communicate with each other. The goal is to avoid any participant bias caused by the help of others. The experimenter is also present in the online meeting room throughout the experimentation to observe participants.

11.3. Materials

Scenarios. We conduct our experiments using two different scenarios to mitigate the effect of the learning curve. We choose the development of *restaurant and library* management systems as our scenarios since they do not require any advanced expertise. Additionally, these scenarios are widely used examples for educational purposes thus eliminating the risk of one group dominating another one. The provided scenarios are publicly available [8].

Pre-experimental survey. We collect general demographic information from the participants (i.e age, gender, occupation, and duration of professional experience). We also asked participants about their source of user story knowledge, whether they only employ user stories for educational purposes or in their professional life, and their level of confidence and understanding while employing user stories.

Table 11.1. Questions for the pre-experimental survey.

Q1	How many years of professional experience working in the IT industry do you have?				
	0	1-3	4-6	More than 6	
Q2	How many times that you participate in constructing user story sets for a software development project? (class projects and assignments)				
	0	1-3	4-6	More than 6	
Q3	How many times that you participate in constructing user story sets for a software development project? (conducted in an agile development cycle)				
	0	1-3	4-6	More than 6	
Q4	Are you actively using user story sets in your daily workflow?				
	1	2	3	4	5
Q5	Which kind of training do you have to learn how to adapt user stories into software development phases?				
	Self study	University course	Company trainings	Industrial experience	I do not have any training on this
Q6	My level of understanding on the theory of user stories				
	1	2	3	4	5
Q7	I feel confident while employing user story sets in my projects.				
	1	2	3	4	5

Table 11.2. Questions for the post-experimental survey.

Strongly Disagree 1	Slightly Disagree 2	Neither Agree Nor Disagree 3	Slightly Agree 4	Strongly Agree 5
This tool helped me to improve the quality of the user stories I write.				
This tool helped me to write more user stories.				
The user interface is user-friendly.				
The suggestions provided by the tool are useful.				
If the tool is available, I would use it for my projects.				

Questionnaire on Effectiveness. We ask participants how effective SCOUT is in terms of increasing user story quality and increasing the number of user stories. We also collect their opinion on whether SCOUT provides a user-friendly interface.

Questionnaire on Suggestion Quality. We asked participants how useful the suggestions that are generated by our system are. We also collect the most and least favorite types of suggestions to determine the level of acceptance of our suggestion strategies.

Questionnaire on Expected Futures. We asked participants about the positive and negative aspects of SCOUT. The responses are collected as free texts and examined by the authors. By asking these questions, we aim to keep our strong aspects whilst improving the weak aspects of our system.

11.4. Study Execution

We conduct our experiment in 70 minutes. We set 30 minutes for each experimental run and additional 10 minutes for the surveys. We split the participants into two groups control and treatment groups to conduct different steps of our evaluation. We use SCOUT along with a collaborative writing editor for measuring the impact of SCOUT. To keep experiment scenarios in their eyesight we put the brief scenario description on the top of the collaborative writing editor and SCOUT's homepage.

For the first run, participants in the control group will be asked to create user stories for the first scenario (S1) by using a collaborative writing editor as a group. Meanwhile, participants in the treatment group will be introduced to SCOUT and asked to use SCOUT to create user stories for the first scenario (S1). After the first run is completed, participants in the control group are introduced to SCOUT for the first time and asked to use SCOUT to create user stories for the second scenario (S2). Meanwhile, participants in the treatment group will be introduced to a collaborative writing editor and asked to use this collaborative writing editor to create user stories for the second scenario (S2). During the experiments with SCOUT, the participants are kindly reminded that SCOUT has a chat functionality for communicating with other members of their group and SCOUT can provide suggestions when requested. After the second run, we will be able to gather data for further calculations to create quantitative

feedback from the participants. To gather qualitative results, participants are asked to fill out our post-experiment survey which consists of questionnaires for effectiveness, suggestion quality, and expected features.

11.5. Results

11.5.1. Demographics

We experiment with a participant group of 24 people who already know user stories. The participant group contains individuals between the ages of 20 and 33. The average age of the participants is 26.1. The age distribution of the participants can be seen in Figure 11.3

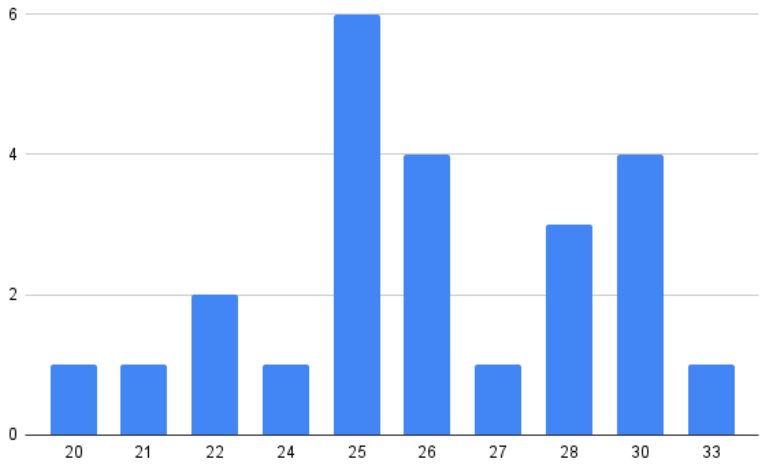

Figure 11.3. Age distribution of the participants.

The participant group has a gender ratio of 70.83% for male participants and 29.17% for female participants. The gender distribution of the participants can be seen in Figure 11.4

We collect the participants' occupations via a demographics survey. 11 of the participants (45.83%) are occupied as software developers. 1 of the participants occupied as a software developer also stated that he/she is an undergraduate student. 2 of the participants occupied as software developers also stated that he/she is a grad-

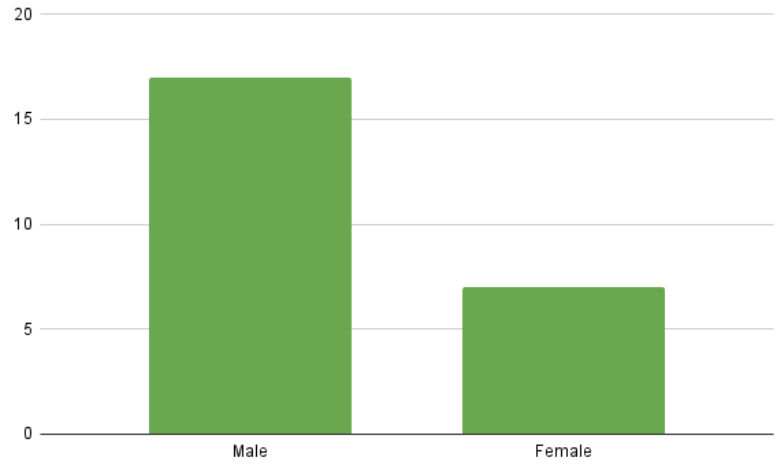

Figure 11.4. Gender distribution of the participants.

uate student. 7 of the participants (29.16%) occupied as data engineers/scientists. 1 of the participants occupied as a data engineer/scientist also stated that he/she is an undergraduate student. 2 of the participants (8.33%) are occupied as electronics engineers/engineers. 1 of the participants (4.16%) is occupied as a risk analyst. 2 of the participants (8.33%) are occupied as undergraduate students. 1 of the participants (4.16%) is occupied as a graduate student. Figure 11.5 presents the occupation distribution of the participants.

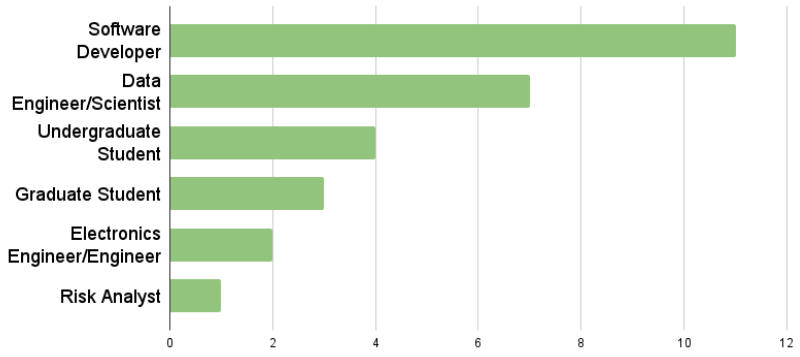

Figure 11.5. Occupation distribution of the participants.

We collect the professional experience that participants have in the IT industry. Results show that 5 of the participants have zero experience in the IT industry. 2 of them are undergraduate students, and 3 of them work outside of the IT industry. Additionally, 14 participants claim that they have 1-3 years of IT industry experience, 3 participants claim that they have 4-6 years of IT industry experience and 2 participants

claim that they have more than 6 years of IT industry experience. Figure 11.6 shows the level of experience that participants have.

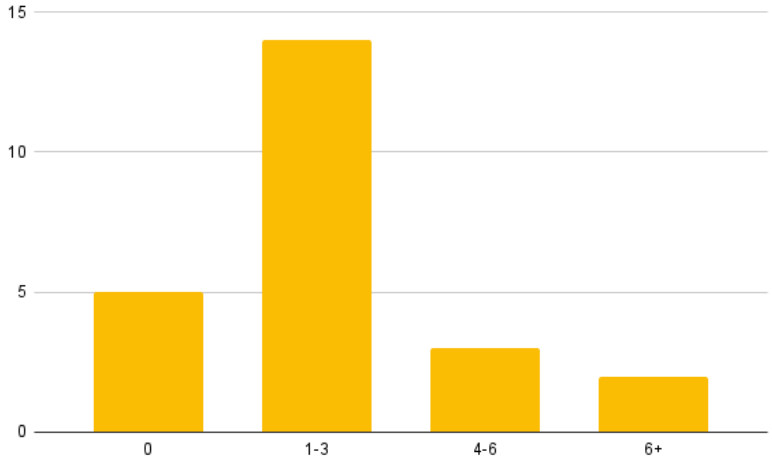

Figure 11.6. IT industry experience levels of the participants.

We also asked different questions to retrieve information about their prior experience with the user stories.

First, we check for their prior experience with user stories in their class projects or assignments. Figure 11.7 represents the participant experience with class projects and assignments. 7 of the participants claim that they did not participate in any class project or assignments that require constructing user stories. 14 of the participants claim that they participated in 1-3 projects. 2 participants claim that they participated in 4-6 projects and 1 participant claim that he/she participated in more than 6 projects.

Second, we check for their prior experience with user stories in an agile development cycle. Figure 11.7 represents the participant experience in an agile development cycle. 12 of the participants claim that they did not participate in any agile development projects or they did not use user stories in an agile development cycle. 7 of the participants claim that they participated in 1-3 projects. 4 of the participants claim that they participated in 4-6 projects and 1 participant claim that he/she participated in more than 6 projects.

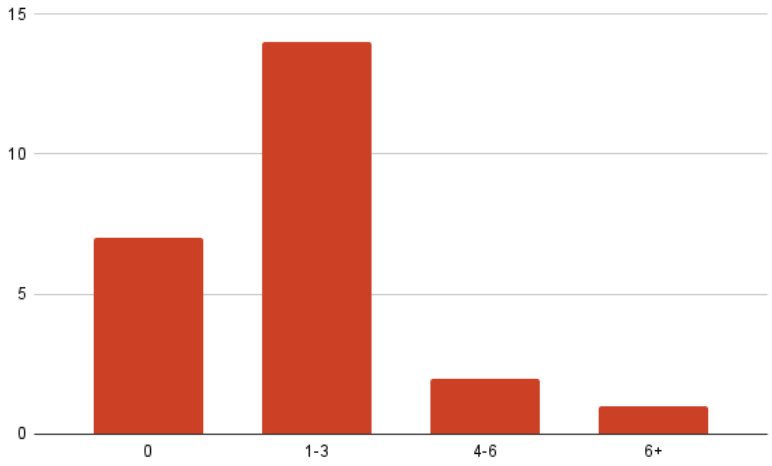

Figure 11.7. Number of participants in constructing user story sets for a software development project (class projects and assignments).

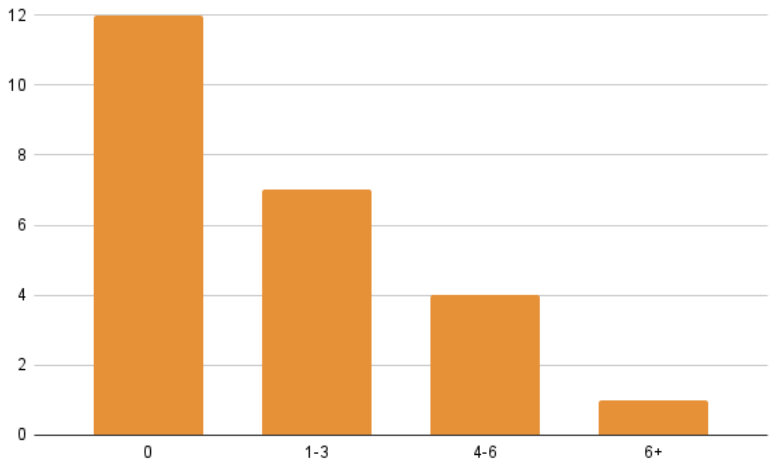

Figure 11.8. Number of participants in constructing user story sets for a software development project (conducted in an agile development cycle).

We also ask the participants whether they use user stories in their daily workflow. The question is asked in a form of a 5-point Likert scale. Figure 11.9 presents the distribution of participants that employ user stories in their daily workflow. 10 of the participants are marked as 3 points or higher and 14 of the participants are marked as 1 or 2. Even though half of the users construct user stories in an agile development cycle, agile development practices seem not to opt for daily workflow for most of the practitioners.

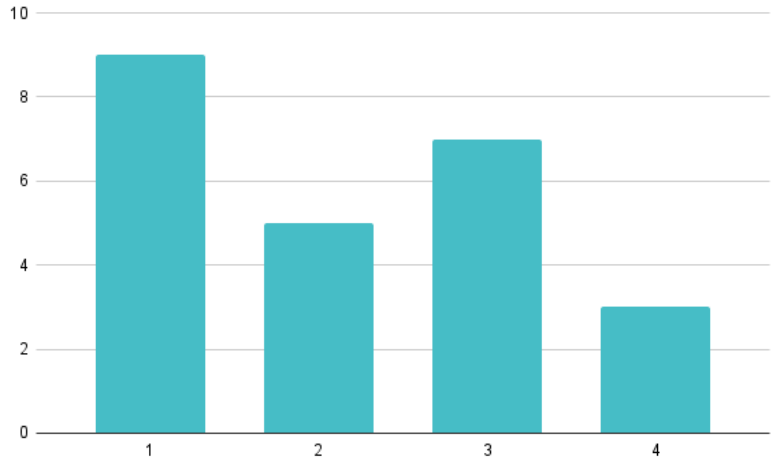

Figure 11.9. Distribution of the participants that actively employ user story sets in their daily workflow.

We also asked the participants about their level of theoretical understanding of user stories. The questions are asked in a form of a 5-point Likert scale. Figure 11.10 presents the levels of understanding of the theory of user stories that participants have. 17 of the participants are marked as 3 points or higher and 7 of the participants are marked as 1 or 2. It shows that the majority of the participants have an adequate understanding of the theory of user stories.

We also asked the participants whether they are confident while employing user stories in their projects. The questions are asked in a form of a 5-point Likert scale. Figure 11.11 presents the levels of confidence in the theory of user stories that participants have. 17 of the participants are marked as 3 points or higher and 7 of the participants are marked as 1 or 2. It shows that the majority of the participants are

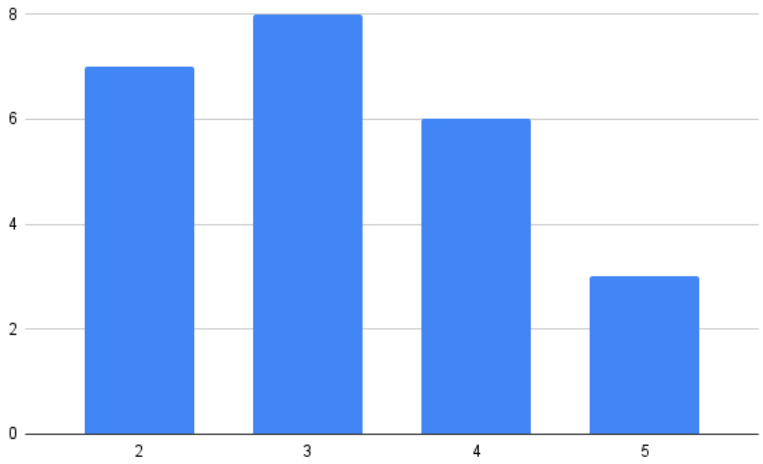

Figure 11.10. Level of understanding of the theory of user stories.

comfortable with user stories.

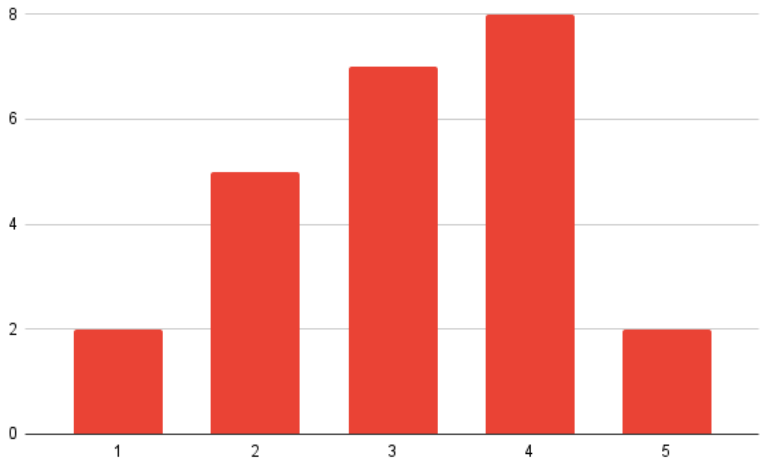

Figure 11.11. Level of confidence while employing user story sets.

We also asked participants for their source of knowledge about user stories. Figure 11.12 presents the source of knowledge about user stories that participants have. 10 of the participants claim that university courses, 4 of the participants claim that industrial experience, 16 of the participants claim that self-study, 9 of the participants claim that company training is their source of user story knowledge, and 3 of the participants claim that they do not have any training on user stories. In general, demographic survey results indicate that our participants have adequate theoretical knowledge and practical experience with user stories.

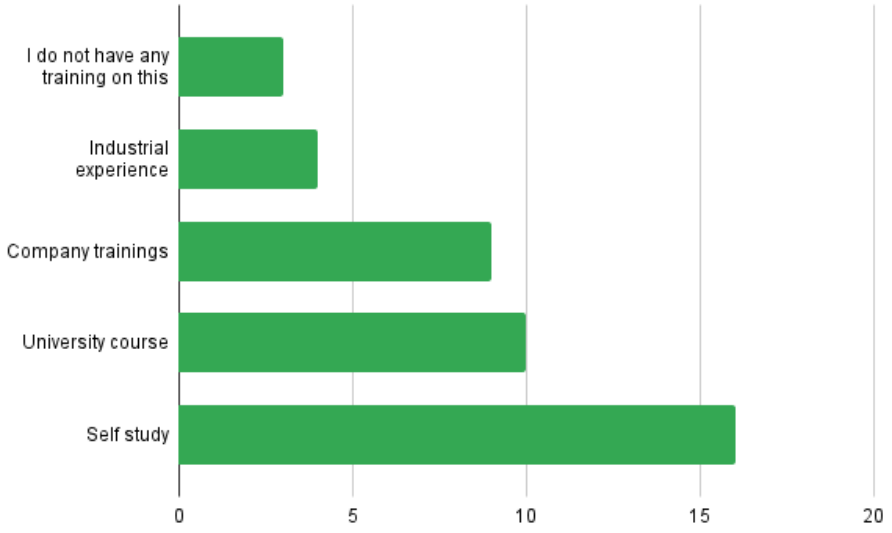

Figure 11.12. Source of knowledge about user stories.

11.5.2. Quantitative Results

RQ3: How effective is our system in terms of increasing the completeness of the user story set?. During our experiment, participants construct 330 user stories using a collaborative writing editor and 448 user stories using SCOUT. Participants manage to construct 35.76% more user stories by using SCOUT. Additionally, we examine each user story that is constructed by participants. We observe an average number of 27.5 user stories per project in the collaborative writing editor. On the other hand, We observe an average number of 37.5 user stories per project with SCOUT.

We also examine the standard deviation of the number of user stories among projects. We observe that the standard deviation of the number of user stories differs significantly. We observe 10.77 for the collaborative writing editor and 9.59 for the SCOUT. To examine this difference in detail, we check whether the samples come from a normal distribution. Results show that number of user stories that are created via using the SCOUT is normally distributed. On the other hand, the number of user stories that are created using a collaborative writing editor is not normally distributed. To check the statistical significance between two independent sample sets, we applied Mann-Whitney U Test since the size of the sample set is relatively small (n = 12 and

$n < 20$). As a result of the Mann-Whitney U test, we obtain a p-value of 0.015 which is smaller than the alpha value of 0.05. Therefore, these results are independent of each other. The difference in variance can also be explained by the concept of writer's block. The lack of assistance to the participants when they need additional ideas causes the number of constructed user stories to remain low. Some of the groups might not face any problem when it comes to constructing user stories continuously however participants with less prior experience with user stories suffer from this problem.

We also examine the graph data. As an initial assumption, we believe that we encourage users to add more definitions to the concepts that they used. When we consider all of the noun phrases as a tree, including different properties of a term yields more branches added to a single node. To examine this assumption, we convert the graphs into trees and apply the breadth-first search starting from the node that has the lowest degree. During the breadth-first search, we collect the number of edges that are included. We observe that graphs that are generated while using SCOUT have 32.46% more edges compared to the user stories that are generated by using a collaborative writing editor. We also observe a slight increase (6.55%) in average node connectivity with SCOUT.

Our null hypothesis H_0 states that completeness for a user story set constructed with an average collaborative text editor is lower or equal to a user story set constructed with SCOUT. However, the results indicate that subjects write significantly more user stories with SCOUT. It can be inferred that SCOUT performs significantly better compared to Google Docs. Therefore, we reject the null hypothesis H_0.

During our experiments, we use two different scenarios as restaurant and library management systems. Hence, we check whether different scenarios influence the results. We hypothesize that H_0: "The scenarios used during experiments have a similar difficulty level". The one-tail alternative hypothesis is H_1: "The scenarios used during experiments have a different difficulty level". To check the statistical significance between two independent sample sets, we applied Mann-Whitney U Test. As a result,

we obtain a p-value of 0.795 which is greater than the alpha value of 0.05. Therefore, we fail to reject H_0. This indicates that the scenarios used during experiments have a similar difficulty level and do not influence the experiment results.

RQ4: To what extent do the participants implement the suggestions produced by SCOUT?. We examine the applied suggestions by participants in two groups as individual and group suggestions. We generate 182 distinct individual suggestions and participants choose to apply 93 (51.10%) of them. We also generate 1262 group suggestions and participants choose to apply 758 (60.06%) of them.

When we look at the number of generated and applied individual suggestions, 5 of the 7 (71.43%) atomic suggestions are applied by participants. A higher portion of applied atomic suggestions yields better quality in individual user stories. Additionally, 88 of 175 (50.29%) isolated suggestions are applied by participants. By generating isolated suggestions, we point out the concepts that need clarification or better explanation so that these concepts are explained in depth. A higher portion of applied isolated suggestions prevents user stories from being ambiguous. The results indicate that our individual suggestion heuristics are accepted by participants and used to increase the individual user story quality.

When we look at the number of generated and applied group suggestions, CRUD suggestions are the least preferred suggestions among others. 57 of the 271 (21.03%) CRUD suggestions are applied by participants. Since we generate additional suggestions for missing CRUD operations, some of them might not be suitable and not employed for each artifact by the participants.

When CRUD suggestions are disregarded, we observe promising results. Participants choose to apply 701 of 991 (70.74%) group suggestions that rely on the difference in main concepts and aim to enlighten different points for users. 76 of 116 (65.52%) close-to-completeness suggestions are applied by the participants. When participants do not face the differences in main concepts, they easily put effort to eliminate the

incompleteness in the user story set. 62 of 83 (74.70%) pop-zero suggestions, 218 of 369 (59.08%) pop-one suggestions, 90 of 109 (82.57%) pop-two suggestions, 216 of 256 (84.38%) pop-three suggestions are applied by the participants. Results show that participants also highly adopt the suggestions that are generated due to the difference in main concepts compared to the whole user story set. The idea behind these suggestions is to enlighten different viewpoints for users to widen their way of thinking.

39 of 58 feeling lucky (67.24%) suggestions applied by participants. We introduce participants to isolated concepts that are created by other users. They use isolated concepts of other users as a reference point to construct user stories that clarify these terms. Therefore, the higher portion of application of these suggestions directly reduces the level of incompleteness in the user story set.

RQ5: What is the perceived usability level?. After the experiment, we conduct a post-experimental survey to collect qualitative results from the participants. We did not obligate participants to attend any part of the experiment. They conduct this experiment voluntarily. Yet, all of them participate in the post-experimental survey and evaluate the usability and suggestion performance of SCOUT.

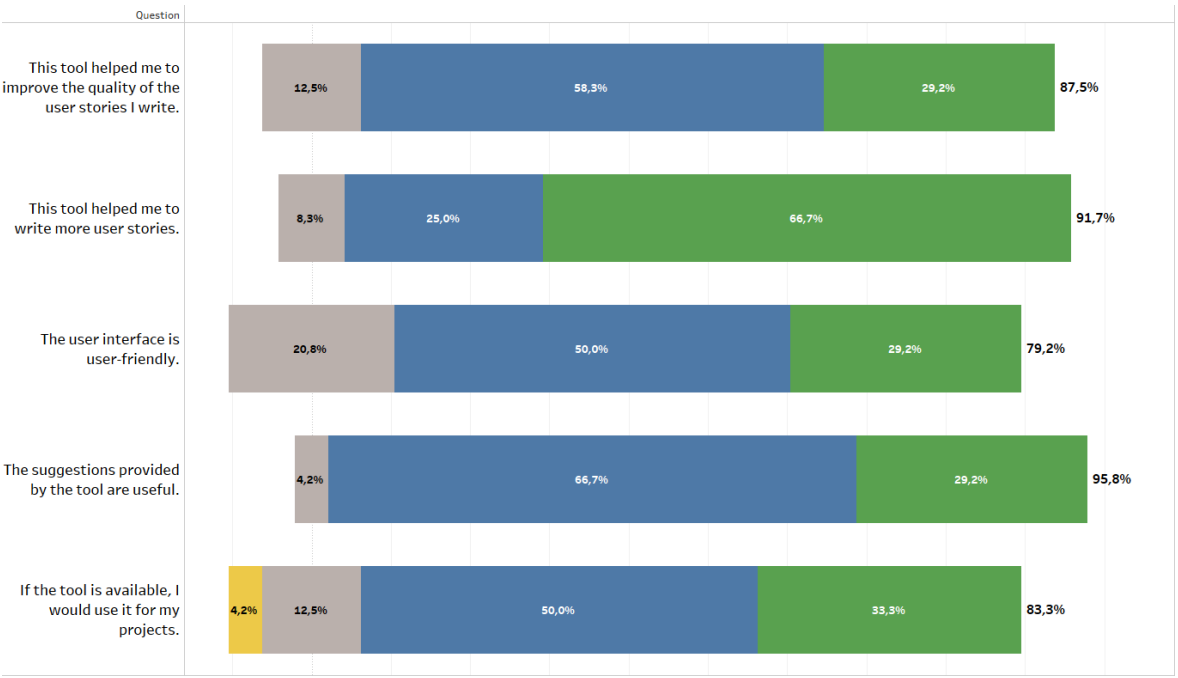

Figure 11.13. Results of the post-experimental survey.

First, we asked participants whether SCOUT improve the quality of the user stories that they wrote during the experiment. This question is asked in a form of a 5-point Likert scale. 3 of them marked 3 points, 14 of them marked 4 points and 7 of them marked 5 points. SCOUT achieves an average point of 4.17 out of 5. We can infer that participants believe that they write better user stories with SCOUT.

We also asked participants whether SCOUT help them to increase the number of user stories. This question is asked in a form of a 5-point Likert scale. 2 of the participants marked 3 points, 6 of them marked 4 points and 16 of them marked 5 points. SCOUT achieves an average point of 4.58 out of 5. We can infer that participants benefit from SCOUT to increase the number of user stories when needed. Quantitative results also show a significantly higher average number of user stories and lower variance in the number of user stories per project. Since participants are randomly assigned, some of them may have more prior experience with the user stories resulting in a difference in the number of user stories among projects. When SCOUT is used, all of the participants have the chance to get support when they need it. Therefore, we can infer that SCOUT helps users to generate more user stories.

We also asked participants whether the user interface of SCOUT is user-friendly. This question is asked in a form of a 5-point Likert scale. 5 of the participants marked 3 points, 12 of the participants marked 4 points and 7 of them marked 5 points. SCOUT achieves an average point of 4.08 out of 5. We can infer that users easily get used to SCOUT without any distractions. While creating the interface of SCOUT, we try to construct a simple user interface. We employ simple components, format instructions, and explanations for suggestions along with functionalities that increase collaboration between users such as a real-time chat function.

We also asked participants whether the suggestions generated by SCOUT are useful. This question is asked in a form of a 5-point Likert scale. 1 of the participants marked 3 points, 16 of the participants marked 4 points and 7 of them marked 5 points. SCOUT achieves an average point of 4.25 out of 5. We can infer that most of the users

benefit from the suggestions. Suggestions are either directly applied to increase quality and completeness or create better ideas in the participants' minds and lead them to generate more comprehensive user stories. As aligned with the quantitative results, a great portion of the suggestions is applied by the participants to enrich the whole user story set.

We also asked participants whether they would use SCOUT for their projects. This question is asked in a form of a 5-point Likert scale. 1 of the participants marked 2 points, 3 of them marked 3 points, 12 of them marked 4 points and 8 of them marked 5 points. SCOUT achieves an average point of 4.13 out of 5. We can infer that most of the users are willing to use SCOUT. Even though being in the early phases of development, SCOUT achieved positive results in terms of acceptance of the approach.

We asked participants to state their favorite features of SCOUT. We collect 24 free text answers from the participants. We categorized them into 2 different groups: suggestion and editor. 21 of the answers indicate that suggestions are by far the most favorite feature of SCOUT followed by 4 answers for the editor. Our main goal with SCOUT is to support stakeholders by generating suggestions and both quantitative and qualitative results reveal that our technique is opted for by participants.

We asked participants to state their most and least useful types of suggestions. We collect 24 answers for each type of evaluation from the participants that contain main suggestion groups (individual and group) and sub-suggestion groups (isolated, atomic, CRUD, etc.). 3 of the participants decide not to share their least type of suggestions. We categorized the answers into two groups individual suggestions, group suggestions, none, and both. 14 of the answers state that one or more of the individual suggestions as their favorite type of suggestions. On the other hand, 14 of the answers state that one or more of the individual suggestions are their least type of suggestion. 11 of the answers state that one or more of the group suggestions are their favorite type of suggestion. On the other hand, 7 of the answers state that one or more of the group suggestions are their least type of suggestion. 6 of the least favorite suggestions

come from feeling lucky types of suggestions. It seems that users have a hard time using that type of suggestion. Combining the results of the applied suggestions and participants' favorite type of suggestion. Group suggestions are widely embraced by the participants. This result is aligned with the quantitative results because a higher portion of the group suggestions is applied by the participants during the experiment.

We also collect participants' positive and negative feedback along with their suggestions to make SCOUT better. we collect 24 positive feedback from the participants that refer to SCOUT as helpful. We collect 15 negative feedbacks to be considered to improve SCOUT. The remainder of the participants does not give any feedback. We categorize the suggestions into 3 different groups suggestions, performance, and editor. 9 of the participants give feedback for improvements in suggestions generated by SCOUT. 1 of the participants is concerned about the non-functional requirement of the performance of the SCOUT. 5 of the participants suggest improvements to increase the usability of SCOUT. We evaluate these suggestions and concerns to form our future work.

11.5.3. Threats to Validity

This section covers the existing threats to SCOUT and, if possible, how those threats have been mitigated. We identify the threats based on the definitions of Wohlin *et al.* [82].

Internal validity. To prevent bias that may arise from using the publicly available user story sets, we design an experiment that requires participants to construct user stories while they are using different tools: SCOUT and Google Docs. Therefore, we are able to mimic a real use-case scenario. We also try to minimize the bias that might arise from participant involvement. Participants are randomly assigned to different groups and the decision for which tool to start is also randomly decided. The order of the available scenarios is also randomly assigned to groups. By doing so, we ensure that equal numbers of randomly assigned groups are introduced to SCOUT in different

experiment sessions with different scenarios. Additionally, we choose the development of *restaurant and library* management systems as our scenarios since they do not require any advanced level of expertise and these scenarios are widely used examples for educational purposes thus eliminating the risk of one group dominating another one. We apply these steps to eliminate any threats caused by the learning effect.

We also consider the needs of the participants while experimenting. First, participants can easily reach our scenario when they are using Google Docs. By default, Google Docs is a text editor which allows them to read the scenario just by scrolling to the top without looking at any other screen or window. To achieve the same accessibility level with SCOUT, we present the scenario on top of the user interface. Second, participants can easily communicate with each other while using Google Docs. Via communication, participants can build consensus on some vague points such as deciding on an actor that is represented as "a manager" or "a restaurant manager" by different stakeholders. To provide a communication channel to users, we implement a message broadcasting mechanism that allows them to communicate with different stakeholders that work on the same project.

External validity. In empirical software engineering research, it is crucial to demonstrate the generalizability of a study. We believe that our participant group is representative of the target user group for the experiment. We form a diverse participant group for the experiments. The participant group involves participants with different backgrounds and levels of expertise. This may potentially reduce concerns about the validity of the study due to the small number of participants as Falessi *et al.* [83] report that a small number of the participants can produce credible results when they accurately represent the target user population. Yet, we seek support from the community to apply and experiment with SCOUT to conclude upon its generalizability.

Construct validity. In agile requirements engineering environments, collaborative writing editors are widely used to construct requirements artifacts. Google Docs provides favorable features for users to ease their work and collaborate easily. We choose

Google Docs which is a widely used collaborative writing editor as a benchmark for SCOUT. This decision is justifiable since Google Docs is a popular tool. Also, employing a generally accepted tool for our experiments reduces the threats caused by participants that are unfamiliar with the collaborative writing editor. After completing our experiments, we choose statistical tests that comply with the obtained data. First, we check whether the samples belonging to different tools come from a normal distribution. Second, we measure the statistical significance of our results by applying Mann-Whitney U Test. Since we collect two independent sets with relatively small sizes, Mann-Whitney U Test is the best fit for our data. We manage to achieve a p-value of 0.015 which is smaller than the alpha value of 0.05. This result indicates that SCOUT obtained a statistically different result compared to Google Docs.

Conclusion validity. Although user studies are inherently subjective and cannot ensure the complete repeatability of results, we try to minimize this issue by sharing our results that are obtained after experiments. To guarantee the consistency of the SCOUT, we established a standardized experimental procedure during the study design and use it for every participant. We also publicly share different materials for replication such as user stories that are generated by participants, and source code for the SCOUT.

12. CONCLUSIONS

This thesis introduces a collaborative requirements editor instrumented with an NLP-powered suggestion functionality. As a component of the NLP pipeline, our study uses a pre-trained deep language model BERT, which has enormous potential for requirements engineering activities. We provide the publicly accessible source code together with information on how the entire system is designed and how it is implemented. Additionally, we contribute to the community by sharing the user stories we gather during the experiments to build a dataset for consequent user story research. We evaluate SCOUT by comparing human performance while using a collaborative text editor and SCOUT. By calculating several metrics and conducting statistical tests, we quantitatively evaluated SCOUT. The quantitative results suggest that by utilizing SCOUT, stakeholders can create user story sets that are noticeably more complete. A statistical significance test supports this inference.

Additionally, for SCOUT offers stakeholders on-demand suggestions while they are creating user stories, participants manage to write more user stories within the experiment period. The findings also demonstrate how eagerly the participants adopted the provided suggestions. The qualitative results suggest that participants are satisfied with SCOUT in a variety of ways. Participants had no problems utilizing SCOUT since they are highly satisfied with the user interface. Additionally, a significant number of users claimed that SCOUT is helpful by offering new ideas to improve the completeness and quality of the user story sets. The majority of the participants also state that if SCOUT is accessible, they will employ it. Overall results suggest that requirements engineering experts would utilize SCOUT. Companies that use agile development principles may simply adopt our technology to their daily workflows since industry awareness of the advantages of requirements engineering practices is continually growing. Comparing this adaption to manual processes may result in lower costs and increased efficiency. We plan to develop the following concerns as future work to carry SCOUT out.

Prevent showing the disliked suggestions. Users can indicate that they dislike some suggestions by clicking the crossed-eye icon in the UI. We collect this feedback to decide on further steps of our implementation to improve the quality of our suggestions. For now, we do not hide these disliked suggestions but we will implement this functionality in the future.

Adaptive recommendation. We plan to implement an intelligent way of providing personalized suggestions for different users. We aim to enable SCOUT to generate suggestions depending on user preferences and behavior to provide a better experience with the system.

Improving the NLP pipeline. In this work, we benefit from the input of the stakeholders in a collaborative project. We aim to expand SCOUT by generating suggestions that contain concepts that stakeholders have never seen before. To do so, we plan to employ another algorithm that retrieves concepts that are related or co-exist with given concepts from a pre-trained deep language model.

Increasing the industry evaluation. Several agile project management tools are used in the IT industry. To increase the industry adoption of SCOUT, we will create a way for integrating SCOUT with agile project management tools such as Jira.

REFERENCES

1. Stallinger, F. and P. Grünbacher, "System Dynamics Modelling and Simulation of Collaborative Requirements Engineering", *Journal of Systems and Software*, Vol. 59, No. 3, pp. 311–321, 2001.

2. Kassab, M., C. Neill and P. Laplante, "State of Practice in Requirements Engineering: Contemporary Data", *Innovations in Systems and Software Engineering*, Vol. 10, No. 4, pp. 235–241, 2014.

3. Kassab, M., "The Changing Landscape of Requirements Engineering Practices Over the Past Decade", *2015 IEEE fifth international workshop on empirical requirements engineering (EmpiRE)*, pp. 1–8, IEEE, 2015.

4. Lucassen, G., F. Dalpiaz, J. M. E. van der Werf and S. Brinkkemper, "The Use and Effectiveness of User Stories in Practice", *International working conference on requirements engineering: Foundation for software quality*, pp. 205–222, Springer, 2016.

5. Dalpiaz, F. and S. Brinkkemper, "Agile Requirements Engineering With User Stories", *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pp. 506–507, IEEE, 2018.

6. Gemkow, T., M. Conzelmann, K. Hartig and A. Vogelsang, "Automatic Glossary Term Extraction from Large-Scale Requirements Specifications", *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pp. 412–417, IEEE, 2018.

7. Aydemir, F. B. and F. Dalpiaz, "Supporting Collaborative Modeling via Natural Language Processing", *International Conference on Conceptual Modeling*, pp. 223–238, Springer, 2020.

8. Köse, S. G. and F. B. Aydemir, "A User Story Dataset for Library and Restaurant Management Systems", https://doi.org/10.5281/zenodo.7529130, 2023, accessed on January 12, 2023.

9. Köse, S. G., "goktugkose/scout: v1.0", https://doi.org/10.5281/zenodo.7529090, 2023, accessed on January 12, 2023.

10. Köse, S. G. and F. B. Aydemir, "Automated Glossary Extraction from Collaborative Requirements Models", *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 11–15, IEEE, 2021.

11. Köse, S. G. and F. B. Aydemir, "A Framework to Improve User Story Sets Through Collaboration", https://arxiv.org/abs/2301.10070, 2023, accessed on January 26, 2023.

12. Hirschberg, J. and C. D. Manning, "Advances in Natural Language Processing", *Science*, Vol. 349, No. 6245, pp. 261–266, 2015.

13. Khurana, D., A. Koli, K. Khatter and S. Singh, "Natural Language Processing: State of the Art, Current Trends and Challenges", *Multimedia tools and applications*, pp. 1–32, 2022.

14. Torfi, A., R. A. Shirvani, Y. Keneshloo, N. Tavaf and E. A. Fox, "Natural Language Processing Advancements by Deep Learning: A Survey", *arXiv preprint arXiv:2003.01200*, 2020.

15. Baeza-Yates, R., B. Ribeiro-Neto *et al.*, *Modern Information Retrieval*, Vol. 463, ACM press New York, 1999.

16. Yu, B., "An Evaluation of Text Classification Methods for Literary Study", *Literary and Linguistic Computing*, Vol. 23, No. 3, pp. 327–343, 2008.

17. Etaiwi, W. and G. Naymat, "The Impact of Applying Different Preprocessing Steps

on Review Spam Detection", *Procedia computer science*, Vol. 113, pp. 273–279, 2017.

18. Jivani, A. G. *et al.*, "A Comparative Study of Stemming Algorithms", *Int. J. Comp. Tech. Appl*, Vol. 2, No. 6, pp. 1930–1938, 2011.

19. Otter, D. W., J. R. Medina and J. K. Kalita, "A Survey of the Usages of Deep Learning for Natural Language Processing", *IEEE transactions on neural networks and learning systems*, Vol. 32, No. 2, pp. 604–624, 2020.

20. Qiu, X., T. Sun, Y. Xu, Y. Shao, N. Dai and X. Huang, "Pre-Trained Models for Natural Language Processing: A Survey", *Science China Technological Sciences*, Vol. 63, No. 10, pp. 1872–1897, 2020.

21. Pennington, J., R. Socher and C. D. Manning, "Glove: Global Vectors for Word Representation", *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

22. Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee and L. Zettlemoyer, "Deep Contextualized Word Representations", *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2227–2237, Association for Computational Linguistics, 2018.

23. Radford, A., K. Narasimhan, T. Salimans and I. Sutskever, *Improving Language Understanding With Unsupervised Learning*, Tech. rep., OpenAI, 2018.

24. Devlin, J., M.-W. Chang, K. Lee and K. Toutanova, "Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding", *arXiv preprint arXiv:1810.04805*, 2018.

25. Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention Is All You Need", *Advances in neural information*

processing systems, Vol. 30, 2017.

26. Wolf, T., L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-Art Natural Language Processing", *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

27. Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov, "Roberta: A Robustly Optimized Bert Pretraining Approach", *arXiv preprint arXiv:1907.11692*, 2019.

28. Sanh, V., L. Debut, J. Chaumond and T. Wolf, "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter", *arXiv preprint arXiv:1910.01108*, 2019.

29. Lan, Z., M. Chen, S. Goodman, K. Gimpel, P. Sharma and R. Soricut, "Albert: A Lite Bert for Self-Supervised Learning of Language Representations", *arXiv preprint arXiv:1909.11942*, 2019.

30. Müller, M., M. Salathé and P. E. Kummervold, "Covid-Twitter-Bert: A Natural Language Processing Model to Analyse Covid-19 Content on Twitter", *arXiv preprint arXiv:2005.07503*, 2020.

31. Vukotic, A., N. Watt, T. Abedrabbo, D. Fox and J. Partner, *Neo4j in Action*, Vol. 22, Manning Shelter Island, 2015.

32. Robinson, I., J. Webber and E. Eifrem, *Graph Databases: New Opportunities for Connected Data*, "O'Reilly Media, Inc.", 2015.

33. Angles, R., "A Comparison of Current Graph Database Models", *2012 IEEE 28th International Conference on Data Engineering Workshops*, pp. 171–177, IEEE, 2012.

34. Lucassen, G., F. Dalpiaz, J. M. E. Van Der Werf and S. Brinkkemper, "Forging High-Quality User Stories: Towards a Discipline for Agile Requirements", *2015 IEEE 23rd international requirements engineering conference (RE)*, pp. 126–135, IEEE, 2015.

35. Dalpiaz, F., I. Van Der Schalk, S. Brinkkemper, F. B. Aydemir and G. Lucassen, "Detecting Terminological Ambiguity in User Stories: Tool and Experimentation", *Information and Software Technology*, Vol. 110, pp. 3–16, 2019.

36. Yang, H., A. De Roeck, V. Gervasi, A. Willis and B. Nuseibeh, "Analysing Anaphoric Ambiguity in Natural Language Requirements", *Requirements engineering*, Vol. 16, No. 3, pp. 163–189, 2011.

37. Barbosa, R., A. Silva and R. Moraes, "Use of Similarity Measure to Suggest the Existence of Duplicate User Stories in the Srum Process", *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pp. 2–5, IEEE, 2016.

38. Yang, H., A. Willis, A. De Roeck and B. Nuseibeh, "Automatic Detection of Nocuous Coordination Ambiguities in Natural Language Requirements", *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pp. 53–62, 2010.

39. Trkman, M., J. Mendling, P. Trkman and M. Krisper, "Impact of the Conceptual Model's Representation Format on Identifying and Understanding User Stories", *Information and software technology*, Vol. 116, p. 106169, 2019.

40. Güneş, T. and F. B. Aydemir, "Automated Goal Model Extraction from User Stories Using NLP", *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pp. 382–387, IEEE, 2020.

41. Wautelet, Y., S. Heng, M. Kolp, I. Mirbel and S. Poelmans, "Building a Rationale

Diagram for Evaluating User Story Sets", *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*, pp. 1–12, IEEE, 2016.

42. Jaqueira, A., M. Lucena, F. M. Alencar, J. Castro and E. Aranha, "Using I* Models to Enrich User Stories", *iStar*, Vol. 13, pp. 55–60, 2013.

43. Wautelet, Y., M. Velghe, S. Heng, S. Poelmans and M. Kolp, "On Modelers Ability to Build a Visual Diagram from a User Story Set: A Goal-Oriented Approach", *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pp. 209–226, Springer, 2018.

44. Teruel, M. A., E. Navarro, V. López-Jaquero, F. Montero and P. González, "CSRML: A Goal-Oriented Approach to Model Requirements for Collaborative Systems", *International Conference on Conceptual Modeling*, pp. 33–46, Springer, 2011.

45. Whitehead, J., "Collaboration in Software Engineering: A Roadmap", *Future of Software Engineering (FOSE'07)*, pp. 214–225, IEEE, 2007.

46. Constantino, K., S. Zhou, M. Souza, E. Figueiredo and C. Kästner, "Understanding Collaborative Software Development: An Interview Study", *Proceedings of the 15th International Conference on Global Software Engineering*, pp. 55–65, 2020.

47. Herbsleb, J. D., "Global Software Engineering: The Future of Socio-Technical Coordination", *Future of Software Engineering (FOSE'07)*, pp. 188–198, IEEE, 2007.

48. Lanubile, F., C. Ebert, R. Prikladnicki and A. Vizcaíno, "Collaboration Tools for Global Software Engineering", *IEEE software*, Vol. 27, No. 2, pp. 52–55, 2010.

49. Konaté, J., A. E. K. Sahraoui and G. L. Kolfschoten, "Collaborative Requirements Elicitation: A Process-Centred Approach", *Group Decision and Negotiation*, Vol. 23, No. 4, pp. 847–877, 2014.

50. Azadegan, A., X. Cheng, F. Niederman and G. Yin, "Collaborative Requirements Elicitation in Facilitated Collaboration: Report from a Case Study", *2013 46th Hawaii International Conference on System Sciences*, pp. 569–578, IEEE, 2013.

51. Hosseini, M., A. Shahri, K. Phalp, J. Taylor, R. Ali and F. Dalpiaz, "Configuring Crowdsourcing for Requirements Elicitation", *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, pp. 133–138, IEEE, 2015.

52. Davis, A., O. Dieste, A. Hickey, N. Juristo and A. M. Moreno, "Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review", *14th IEEE International Requirements Engineering Conference (RE'06)*, pp. 179–188, IEEE, 2006.

53. Van Lamsweerde, A., "Requirements Engineering in the Year 00: A Research Perspective", *Proceedings of the 22nd international conference on Software engineering*, pp. 5–19, 2000.

54. Dalpiaz, F., A. Ferrari, X. Franch and C. Palomares, "Natural Language Processing for Requirements Engineering: The Best Is Yet to Come", *IEEE software*, Vol. 35, No. 5, pp. 115–119, 2018.

55. Zhao, L., W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca and R. T. Batista-Navarro, "Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study", *arXiv preprint arXiv:2004.01099*, 2020.

56. Ferrari, A. and A. Esuli, "An NLP Approach for Cross-Domain Ambiguity Detection in Requirements Engineering", *Automated Software Engineering*, Vol. 26, No. 3, pp. 559–598, 2019.

57. Duan, C., P. Laurent, J. Cleland-Huang and C. Kwiatkowski, "Towards Automated

Requirements Prioritization and Triage", *Requirements engineering*, Vol. 14, No. 2, pp. 73–89, 2009.

58. Robeer, M., G. Lucassen, J. M. E. Van Der Werf, F. Dalpiaz and S. Brinkkemper, "Automated Extraction of Conceptual Models from User Stories via NLP", *2016 IEEE 24th international requirements engineering conference (RE)*, pp. 196–205, IEEE, 2016.

59. Hotomski, S. and M. Glinz, "GuideGen: An Approach for Keeping Requirements and Acceptance Tests Aligned via Automatically Generated Guidance", *Information and Software Technology*, Vol. 110, pp. 17–38, 2019.

60. Stanik, C., M. Haering and W. Maalej, "Classifying Multilingual User Feedback Using Traditional Machine Learning and Deep Learning", *2019 IEEE 27th international requirements engineering conference workshops (REW)*, pp. 220–226, IEEE, 2019.

61. Hey, T., J. Keim, A. Koziolek and W. F. Tichy, "NoRBERT: Transfer Learning for Requirements Classification", *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pp. 169–179, IEEE, 2020.

62. de Araújo, A. F. and R. M. Marcacini, "RE-BERT: Automatic Extraction of Software Requirements from App Reviews Using BERT Language Model", *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pp. 1321–1327, 2021.

63. Kujala, S., "User Involvement: A Review of the Benefits and Challenges", *Behaviour & information technology*, Vol. 22, No. 1, pp. 1–16, 2003.

64. Kujala, S., M. Kauppinen, L. Lehtola and T. Kojo, "The Role of User Involvement in Requirements Quality and Project Success", *13th IEEE International Conference on Requirements Engineering (RE'05)*, pp. 75–84, IEEE, 2005.

65. El Emam, K., S. Quintin and N. H. Madhavji, "User Participation in the Re-

quirements Engineering Process: An Empirical Study", *Requirements engineering*, Vol. 1, No. 1, pp. 4–26, 1996.

66. Snijders, R., F. Dalpiaz, S. Brinkkemper, M. Hosseini, R. Ali and A. Ozum, "REfine: A Gamified Platform for Participatory Requirements Engineering", *2015 IEEE 1st International Workshop on Crowd-Based Requirements Engineering (CrowdRE)*, pp. 1–6, IEEE, 2015.

67. Snijders, R., F. Dalpiaz, M. Hosseini, A. Shahri and R. Ali, "Crowd-Centric Requirements Engineering", *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pp. 614–615, IEEE, 2014.

68. Deterding, S., D. Dixon, R. Khaled and L. Nacke, "From Game Design Elements to Gamefulness: Defining "Gamification"", *Proceedings of the 15th international academic MindTrek conference: Envisioning future media environments*, pp. 9–15, 2011.

69. Fernandes, J., D. Duarte, C. Ribeiro, C. Farinha, J. M. Pereira and M. M. da Silva, "IThink: A Game-Based Approach Towards Improving Collaboration and Participation in Requirement Elicitation", *Procedia Computer Science*, Vol. 15, pp. 66–77, 2012.

70. Cohn, M., *User Stories Applied: For Agile Software Development*, Addison-Wesley Professional, 2004.

71. Wake, B., "Invest in Good Stories and Smart Tasks, August 2003", *URL http://xp123. com/articles/invest-in-good-stories-and-smart-tasks*, 2003.

72. Lucassen, G., F. Dalpiaz, J. M. E. van der Werf and S. Brinkkemper, "Improving Agile Requirements: The Quality User Story Framework and Tool", *Requirements engineering*, Vol. 21, No. 3, pp. 383–403, 2016.

73. Wieringa, R. J., *Design Science Methodology for Information Systems and Software*

Engineering, Springer, 2014.

74. Doe, J., "Recommended Practice for Software Requirements Specifications (IEEE)", *IEEE, New York*, 2011.

75. Heck, P. and A. Zaidman, "A Quality Framework for Agile Requirements: A Practitioner's Perspective", *arXiv preprint arXiv:1406.4692*, 2014.

76. "spaCy", https://spacy.io/, accessed on January 12, 2023.

77. "textacy: NLP, before and after spaCy", https://textacy.readthedocs.io/en/latest/, accessed on January 12, 2023.

78. "Hugging Face", https://huggingface.co/, accessed on January 12, 2023.

79. Reimers, N. and I. Gurevych, "Sentence-Bert: Sentence Embeddings Using Siamese Bert-Networks", *arXiv preprint arXiv:1908.10084* , 2019.

80. Grootendorst, M., "KeyBERT: Minimal Keyword Extraction With BERT.", https://doi.org/10.5281/zenodo.4461265, 2020, accessed on January 12, 2023.

81. "Neo4j Bolt Driver for Python", https://github.com/neo4j/neo4j-python-driver, accessed on January 12, 2023.

82. Wohlin, C., P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering*, Springer Science & Business Media, 2012.

83. Falessi, D., N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka and M. Oivo, "Empirical Software Engineering Experts on the Use of Students and Professionals in Experiments", *Empirical Software Engineering*, Vol. 23, No. 1, pp. 452–489, 2018.