Automated Software Engineering (2023) 30:7
https://doi.org/10.1007/s10515-022-00371-9

An NLP-based quality attributes extraction and prioritization framework in Agile-driven software development

Mohsin Ahmed[1] · **Saif Ur Rehman Khan**[1] · **Khubaib Amjad Alam**[2]

Received: 1 December 2020 / Accepted: 7 October 2022 / Published online: 9 January 2023

Abstract

Software quality plays a significant role in ensuring the customer demands and expectations. Generally speaking, Quality of the software is a functional behaviour that heavily depends on the non-functional requirements. However, generally software engineer's pay relatively lesser attention to the non-functional requirements. Moreover, it is of vital importance to have a clear view of software's quality as early as possible, because it can affect the different artefacts of the software at later development stages including implementation, testing, and maintenance. The early-stage conformance of software quality is more important in agile-based software development where the requirements are more volatile than any other development environments. The early knowledge about the software quality can positively impact on the design decisions in agile-based software development context. Motivated by this, we propose a conceptual framework for automatic extraction and prioritization of quality attributes from the user stories in an agile-based development context. The proposed framework contains two main components including QAExtractor and QAPrioritiser. The core of this framework (QAExtractor) is based on natural language processing, which generalise the user stories for a specific quality attribute. In contrast, QAPrioritiser ranks the extracted quality attributes grounded on the frequency, roles impact, and criticality factor value. We validate the effectiveness of the proposed framework using two case studies. The results revealed that the proposed framework outperforms the existing technique in terms of precision, recall, and F measure.

Keywords Software quality attributes · NFRs requirements prioritization · Agile-based software development · Software quality assurance

✉ Khubaib Amjad Alam
khubaib.amjad@nu.edu.pk

Extended author information available on the last page of the article

1 Introduction

Software quality plays a vital role in ensuring the customer's demands and expectations. To find the user's requirements of a targeted system, requirement engineering is widely used by the software organisations. Generally, requirement engineers focus on the Functional Requirements (FRs) and neglect the Non-Functional Requirements (NFRs) during the requirements elicitation process. Consequently, this negligence lacks in ensuring the expected behaviour of the software, which may lead to the software failure (Domah 2013; Maiti et al. 2017). The NFRs play an important role while taking design decision (Chung et al. 2012; Rehman et al. 2021). The quality of software goes throughout its development, i.e. from requirements elicitation to its deployment and maintenance. The software quality depicts the functional behaviour of software by representing the customer's satisfaction level with the provided functionality. Due to this fact, the software quality attributes are strongly bonded to the FRs (Moreira et al. 2002; Al Imran et al. 2017). The quality attributes of software are the properties that affect the software as an individual or whole. Besides NFRs, software quality attributes also depend on the domain or context of the software and specific development phase of Software Development Life Cycle (SDLC) (Arvanitou et al. 2017). The required quality level of the software is an obstacle in its development because it adds extra work to the actual functionality (Hneif and Lee 2010).

The requirements are written or recorded as user stories in Agile-based Software Development(ASD) context (Kassab 2015; Schön et al. 2017). The factor of uncertainty becomes high especially in the case of user stories as the stakeholders are allowed to ask for the changes at anytime. Consequently, it makes the situation more difficult to decide about the quality attributes in the context of ASD. There are some reasons due to which the specification of quality attributes becomes challenging. First of all, the Requirements Elicitation Team (RET) lacks in paying required level of attention to the quality aspect of the software during requirements elicitation phase in an ASD (Chung and Prado Leite 2009). Secondly, the quality attributes can vary for various groups of the stakeholders. Even a single user can have a different view of the quality attributes. For example, one user can consider the quality attribute "A" as most important, while another user can consider it as least important among all other found quality attributes (Etxeberria 2007). Hence, the quality attributes need to be prioritized in order to clarify their importance in a specific software context. Notice that some quality attributes are explicitly defined such as usability, security, functionality, and so on. While, other quality attributes can be implicitly described like efficiency, learn-ability, and so on.

There might be a scenario in which a conflict can occur among the quality attributes that can disturb the whole design decisions and it can also lead the software system to the failure (Al Imran et al. 2017). The view of software's quality will only be cleared if a set of relevant quality attributes are identified. Once the quality attributes are correctly identified, the design engineers can effectively perform appropriate design decisions with a great level of confidence. This would

ultimately minimise the rework effort, which could occur in the case of software failure. So, it is essential to identify the appropriate set of quality attributes, and then prioritise them according to a prioritization mechanism. Hence, there is a need for a framework capable to automatically extract the quality attributes, and prioritise the identified quality attributes according to their importance in ASD context.

In this research work, we propose a framework facilitating the automatic extraction and prioritization of the quality attributes in ASD context. The proposed framework is comprised of two main components: (i) QAExtractor to extract quality attributes from the user stories, and (ii) QAPrioritiser to rank the extracted quality attributes based on the devised prioritization mechanisms. The QAExtractor is grounded on Natural Language Processing (NLP), which is a linguistic and cognitive science approach and analyse many natural language data (Collobert et al. 2011). As the user requirements (user stories in case of ASD) are described in the natural language (Ferrari et al. 2017), NLP is more suitable to parse the user stories for identification and prioritization of the quality attributes. The main contributions of this research work are listed as follows:

1. Devise and implement a conceptual framework for quality attributes extraction - QAExtractor: Automatically extracting the quality attributes from Agile user stories using Natural Language Processing at early development stage of SDLC (i.e. requirements engineering).
2. Develop a ranking technique along with three mechanisms for prioritization of the quality attributes - QAPrioritiser: Automatically prioritising the extracted quality attributes to define their role and impact on the targeted software.
3. Outlines a set of future directions for the researchers and practitioners.

2 Literature review

In the literature, limited research studies have been reported that aimed to identify or extract quality attributes of the software project especially at the early stages of SDLC. We have conducted a extensive literature review and analysed current state-of-the-art quality attributes extraction techniques in ASD context. The techniques can be classified into two main groups: (i) Algorithmic, and (ii) Non-Algorithmic techniques. The following sections describe the reported algorithmic and non-algorithmic quality attributes' extraction techniques.

2.1 Algorithmic QAE techniques

This section provides the algorithmic QAE techniques. Farid (2012) presented a methodology effective for modelling of NFRs, named NORMAP. The author specifically tailored the NFRs modelling framework for Agile practices. They used Chung's NFRs framework (Chung et al. 2012) for linking NFRs with the FRs in the form of coded colours, where they represented FRs as Agile Use Cases (AUC) and

NFRs as Agile Loose Cases (ALC). However, this is only feasible when both, the functional and non-functional requirements, are elicited. They reported that NORMAP achieved the classification success of 87.15% and their framework can be used as a manual or semi-automated, i.e. NFRs Modelling for Agile Manual (NORMANUAL) and NFRs Modelling with Agile Automatic (NORMATIC) (Farid and Mitropoulos 2012). However, this framework lacks in mapping NFRs with FRs at the requirements elicitation phase. Moreover, NORMAP focused on the designing phase of SDLC, which may required significant amount of rework effort.

Gilson et al. (2019) presented a study in which they investigated whether the user stories contain information about quality attributes or not. The authors concluded that several quality attributes, including compatibility, security, usability, maintainability, performance, can be extracted from the user stories. They utilised a Machine Learning (ML) based classifier using the SpaCy library that is an extension of Google's universal part-of-speech tag-set (Petrov et al. 2011). The base of SpaCy library is on Convolutional Neural Network (CNN) (Yin et al. 2017), and is reliable in terms of performance and learn-ability (Al Omran and Treude 2017). First, they implemented one global model trained for all quality attributes. Second, they applied specialised models trained for individual quality attributes. The experimental results showed that the global model outperformed the specialised models. However, the authors only tested compatibility and maintainability attributes. Furthermore, the model requires incremental training to obtain better accuracy in terms of quality attributes' extraction, which remains a time-consuming process.

Knauss and Ott (2014) modelled a socio-technical system for the classification of natural language requirements. The authors compared the performance of semi-automatic, fully automatic, and manual system, and found that the semi-automatic outperforms other compared techniques. They reported that maximum precision of semi-automatic version is 0.91. However, the test result's depict that the obtained performance is not consistent as it varies from 0.57 to 0.91. Moreover, their proposed technique requires more time than the automatic technique because its learning part is heavily dependent on the experts due to requirements classifying.

Lu and Liang (2017) adopted a vector-based classification technique for the extraction of NFRs from augmented app user reviews. The authors classified NFRs in four types, including performance, portability, reliability, and usability, using Naive Bayes, J48, and Bagging. They reported to achieve the precision of 71.4%. However, the proposed technique is only feasible where a relevant app is already available, and it contains a number of reviews. Moreover, the authors only targeted four quality attributes, and lacks in classifying the remaining quality attributes such as security, integrity, synthesis, ease of access, and so on.

In comparison to the discussed algorithmic techniques,Kurtanović and Maalej (2017) leveraged Support Vector Machine (SVM) based binary classifier to classify requirements as FRs and NFRs. Furthermore, the authors employed multi-class NFR classifier to classify NFRs. Their test results showed the precision of 92% for both classifications. However, the precision is only achieved with the oversampling of 100%. Moreover, they only classified NFRs into four types, including operational, performance, usability, and security, and lacks in identifying/ extracting rest of the quality attributes.

In contrast, Abad et al. (2017) classified NFRs by using NLP (parts of speech tagging, entity tagging, and temporal tagging). They assigned weights to the more occurring words in the user stories by using co-occurrence and regular expressions. After that they used Decision Tree (DT) for the classification of NFRs. Moreover, they used reviews of 40 top rated apps from Amazon. However, their proposed technique requires iterative removal of encoding and formatting errors. Regarding the results, the authors mentioned that the proposed technique performed well in classifying the requirements into FRs and NFRs. However, the proposed technique lacks in sub-classifying NFRs, which requires additional sentimental structures/ patterns.

2.2 Non-algorithmic QAE techniques

This section provides the non-algorithmic QAE techniques. Jeon et al. (2011) proposed a quality attributes-driven agile development method named as ACRUM, complying with the core activities of the SCRUM. Their proposed model incorporated the aspect of NFRs (i.e. quality attributes) along with the FRs which were missing in the SCRUM process due to its flexible nature. The FRs were analysed to maintain the product backlog through which the quality attributes were mapped. After implementing the sprint backlog, they verified FRs and quality attributes by a demonstration.

Bellomo et al. (2013) focused on the quality attributes of the software by incremental prototyping where the requirements were evolving. Firstly, the feature was developed, and then early feedback was taken from the relevant stakeholder after the sprint. An architectural trade-off is designed and discussed with the stakeholder at the post-user demo, and if the stakeholder agreed on the functionality and quality of the feature, then it is approved, otherwise rework is being done. Later, the prototype-module was integrated with the actual project at release planning.

In contrast, Aljallabi and Mansour (2015) integrated two existing approaches for the analysis of NFRs, and proposed an enhanced approach to NFRs analysis. The authors considered both perspectives of NFRs, including internal and external quality attributes, to guarantee their correct identification. Their proposed approach comprised of six phases. In phase 1, they elicited FRs from the users. In phase 2, they prepared a mapping sheet between FRs and external quality attributes where the developer used his technical knowledge to include and exclude appropriate external quality attributes according to the elicited FRs. In phase 3, the sheet is handed over to the customer and asked to map each FR with the required quality attributes. In phase 4, internal quality attributes are decided between customer and project members. In phase 5, a mapping sheet between external and internal quality attributes is created to remove the conflict between them. Finally, this mapping sheet is discussed with the customer and finalised the attributes after the discussion.

Overall comparison of current state-of-the-art algorithmic and non-algorithmic QAE techniques is shown in Table 1. The comparison is grounded on the parameters identified from the literature.

From Table 1, it can be observed that one of the most basic purposes of algorithmic techniques is to achieve the automation. The non-algorithmic heavily depends

Table 1 A comparative analysis of existing algorithmic and non-algorithmic QAE techniques

Parameter	QAE Techniques	
	Algorithmic	Non-algorithmic
Automation	✓	✗
Need of experts	✗	✓
Tacit knowledge	✗	✓
Early phase detection	✗	✗
Reduced time complexity	✓	✗
Historical data	✓	✗
Incremental training	✓	✗

upon the experts and their tacit knowledge, which they gain along with their experience. In contrast, this intelligence is embedded in the algorithms to perform the tasks of these experts. As a result, it significantly decreases the overall time complexity required to solve a problem. This is because of the fact that the processing speed of a human is much slower than the computer. However, the algorithmic techniques require historical data to obtain this intelligence. The algorithmic techniques incrementally train themselves to achieve maximum performance.

2.3 Quality attributes prioritization

This section describes the reported work related to quality attributes prioritization. In the literature, only two works (Jawad and Bashir 2015; Jain et al. 2016) have been mentioned that focused on ranking the quality attributes. Jawad and Bashir (2015) developed an inter-relationship among quality attributes. Finally, the authors qualitatively prioritized the quality attributes based on the power and dependency among them. In contrast, Jain et al. (2016) used Interpretative Structural Modelling (ISM) and presented a framework to model and measure the quality attributes in such a way that most critical attributes can be determined. However, the identification of quality attributes was made with the subjective analysis of experts, i.e.through expert-opinion based technique.

3 Proposed framework

This section presents the proposed conceptual framework, effective for automatic extraction and prioritization of software quality attributes from the user stories in ASD context. The proposed framework is comprised of two major components including QAExtractor and QAPrioritiser as depicted in Fig. 1. Notice that QAExtractor component is responsible for automatic extraction of software quality attributes from the user stories. In comparison to QAExtractor, QAPrioritiser

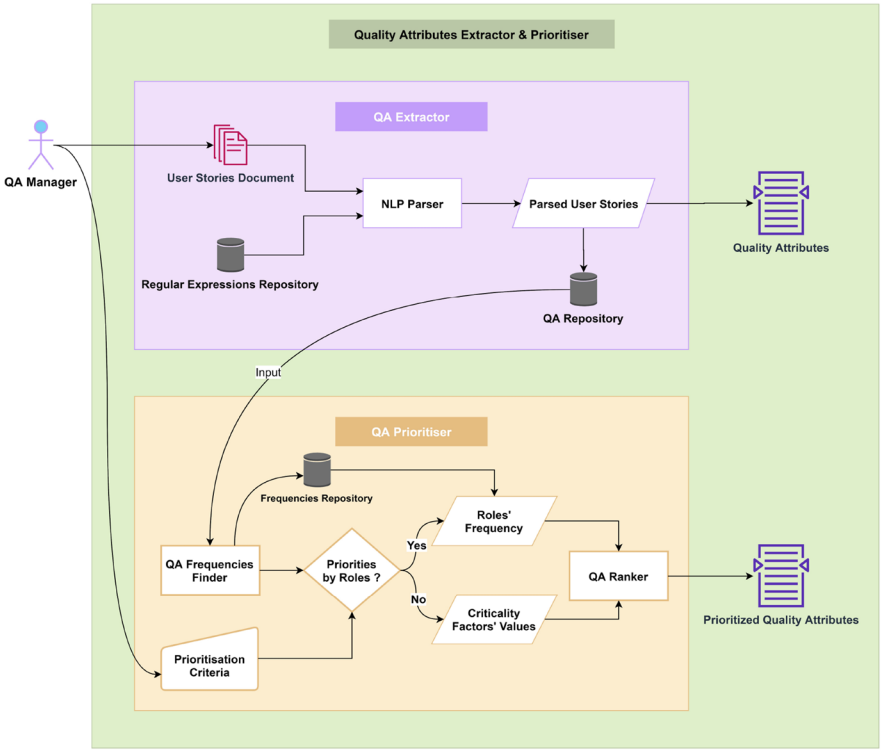

Fig. 1 Proposed conceptual framework

is responsible for the ranking of extracted quality attributes. The following sections provide the details regarding each of the components of the proposed framework.

3.1 QAExtractor

QAExtractor extracts the quality attributes from the user stories in the context of ASD. The input of this component is user stories document, as provided by the Quality Assurance (QA) Manager. NLP is the core of this component, which is based on regular expressions. The main underlying reason of employing NLP is that the user stories are written in natural language format. Moreover, the user stories are less formal and represents a complete story as mentioned by the user. We are leveraging the Regular Expressions in this work where first of all, specific keywords are tokenized. Next, their context is analysed. Notice that regular expression are used for the context analysis. If the context and tokenized keywords are matched, then we can easily extract the quality attribute. By doing this, we are avoiding the overlapping of quality attributes.

3.1.1 User stories

User stories are the FRs of the software that are stated by the user. Although, the user stories are not directly stated to specify the quality of software. However, they have some hidden indications that could be useful in indicating the quality. The user stories are passed to the Natural Language Processor to scan them in a sequential manner, and then finds the hidden quality attribute. The quality attributes could be either explicitly stated or not. The user story doesn't need to be in a specific format for ASD such as "As a ⟨ type of user⟩, I can ⟨ perform some task⟩, so that I can ⟨ achieve some goal⟩". Notice that the proposed technique is specifically designed for ASD context. Thus, the validation module of the Natural Language Processor only validates the user story written in a specific format.

3.1.2 Regular expression

Regular expressions are the core part of QAExtractor component of the proposed framework. Actually, they act like a search patterns used to search and identify the specific keywords in the user stories. Notice that we cannot directly search these keywords because they are compound keywords and contain more than one orders. Moreover, a word could have multiple synonyms and could be in different forms such as verb, noun, and so on. So, there is a need to design regular expressions for different quality attributes to precisely extract them from the user stories. Naturally, there could be more than one regular expressions for one quality attributes as it could be expressed in different ways. In this work, we are using two sets of regular expressions: one for the identification of quality attributes from the user stories, and second for the identification of role (stakeholder) of the particular user story.

3.1.3 NLP-parser

NLP-Parser is responsible to fetch the regular expressions and parse the user stories in order to identify their syntactic structure. If the user story successfully parses through the regular expression, then it fetches the context and corresponding quality attribute of a user story and stores it. Algorithm 1 presents the pseudo-code parsing user stories document and producing the extracted quality attributes.

Algorithm 1: User stories Parsing Algorithm

```
Input: User Stories Document
Result: Quality Factors, Quality Attributes
extractedQualityAttributes = null;
userStories = scanUserStories(userStoriesDocument);
for userStories 1 to m do
    userStory = userStories[m];
    for regularExpressions 1 to n do
        regularExpression = regularExpressions[n];
        if parse(userStory, regularExpression) === true then
            extract quality attribute, i.e. the index name of the regular
              expression and store it in an array;
            qualityAttribute = indexOf(regularExpression);
            push qualityAttribute into extractedQualityAttributes;
        else
            continue;
        end
    end
end
Return: extractedQualityAttributes;
```

The input of Algorithm 1 is the user stories' document written in the form of natural language. The algorithm will parse all user stories one by one. As there is possibility that a user story can contain more than one quality attributes, the user story will be parsed through all defined rules, i.e., regular expressions. These rules are stored in the repository, from where they are fetched into the array of **regularExpressions**. Once, a user story is successfully parsed through the rule and a quality attribute is identified successfully, it'll be stored in to the array of **extractedQualityAttributes**. At the end, the whole array will be returned as an output of the algorithm.

3.2 QAPrioritiser

This section provides the working mechanism of QAPrioritiser component. Once the user stories are parsed and the appropriate quality attributes are successfully extracted, QAPrioritiser ranks the extracted quality attributes. The input of QAPrioritiser component is the parsed user stories along with the quality attributes and the roles (stakeholders). The frequency finder computes the frequency of extracted quality attributes and roles, and passes them to the QARanker (core of this component). The QARanker ranks the quality attributes grounded on three prioritization criterion: (i) based on the quality attributes' frequency, (ii) by aggregating quality attributes frequency along with the frequency of roles, and (iii) based on the

Criticality Factor Value (CFV). Finally, the prioritized quality attributes are stored in QA Repository (Fig. 1).

3.3 Quality attributes extraction (QAE) method

This section provides the details about QAE method. A flow chart of QAE method is depicted in Fig. 2. The QAE is initiated with the input of user stories' document. As soon as the document is provided, it parses the whole document, gets the user stories, and excludes the rest of the data. The regular expressions are designed to verify either the user stories are in the standard format or not. Once, the user stories are validated to be the standard format, they are collected from the document.

For example, a user story with a standard format is shown in Fig. 3a. The regular expression for it's validation is also represented in Fig. 3b.The output after parsing user story through the regular expression is also shown in Fig. 3c. The standard structure for a user story is highlighted to illustrate the parsing results. In this work, We slightly modify the regular expression to accommodate different articles such as a, an, the and pronouns including I, we.

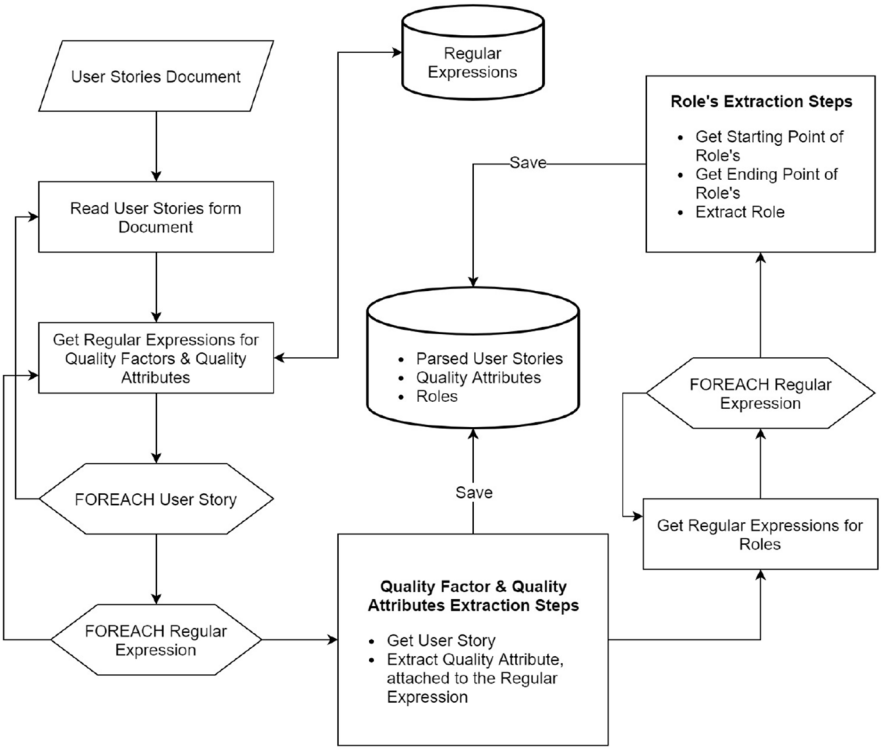

Fig. 2 Quality attributes extraction method

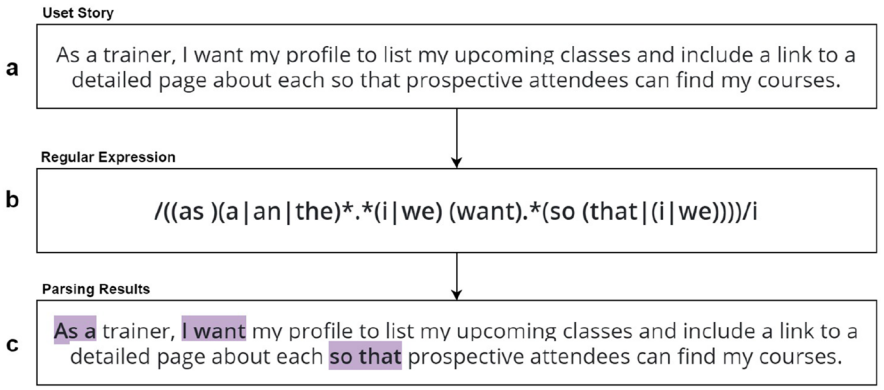

Fig. 3 Validation of a user story

After getting all user stories from the document, the extraction method gets regular expressions to extract quality attributes from the user stories. It reads the collected user stories one by one, parses each user story through all regular expressions, and checks whether the user story is successfully parsed through any regular expression or not. The regular expressions are stored in the form of a multi-dimensional associative array and the names of the index of this array are the quality factors and quality attributes as shown in Fig. 4. Once, the user story is successfully parsed through the regular expression, it extracts the attached quality factor and quality attribute to the regular expression, and stores them. When all user stories are successfully checked against all regular expressions, the extracted quality factors and quality attributes are gathered and organised to design a quality model.

```
Regular Expressions for Quality Attributes = array(
    'security'    =>    array(
        'privacy'     =>    'regular expression',
        'integrity'    =>    'regular expression',
        .,
        .,
        .
    ),
    'useability' =>    array(
        'ease of access'    =>    'regular expression'
    ),
    .,
    .,
    .
);
```

Fig. 4 Regular expressions stored in a multi-dimensional associative array

A working illustration of QAExtractor is shown in Fig. 5. As a sample, a user story (i.e. from the dataset used for the experimentation) is shown in Fig. 5a. A multi-dimensional associative array of regular expressions is shown in Fig. 5b. For the illustration purpose, only one regular expression is shown. Finally, the parsing results of a user story are shown in Fig. 5c. The phrase that successfully parsed through the given regular expression is also highlighted. Since the user story is successfully parsed through the regular expression, the indexes of the associative array are collected as quality attributes including useability and ease of access.

3.4 Quality attributes prioritization method

In this work, we devise three types of quality attributes Prioritization mechanisms including prioritization by frequency of quality attributes, prioritization by aggregating quality attributes' frequency with the frequency of roles, and prioritization by CFV of quality attributes. The quality attributes' frequency refers to the number to which extent an attribute is extracted from all of the inputted user stories. For example, a quality attribute "integrity" is found 5 times in all given user stories, then the frequency of integrity is 5. In contrast, roles' frequency is the number of user stories stated by a specific role. In comparison to quality attributes' and roles' frequency, CFV is the value of a quality attribute that a quality assurance (QA) manager defines. This could vary from project to project because the criticality of one quality attribute could be different in different projects and contexts. A flow chart for finding frequencies of quality attributes and roles is shown in Fig. 6.

The parsed user stories along with the quality attributes and roles are passed to the frequencies finder. All linked user stories with the quality attributes are processed one by one to calculate the frequencies of linked quality attributes and

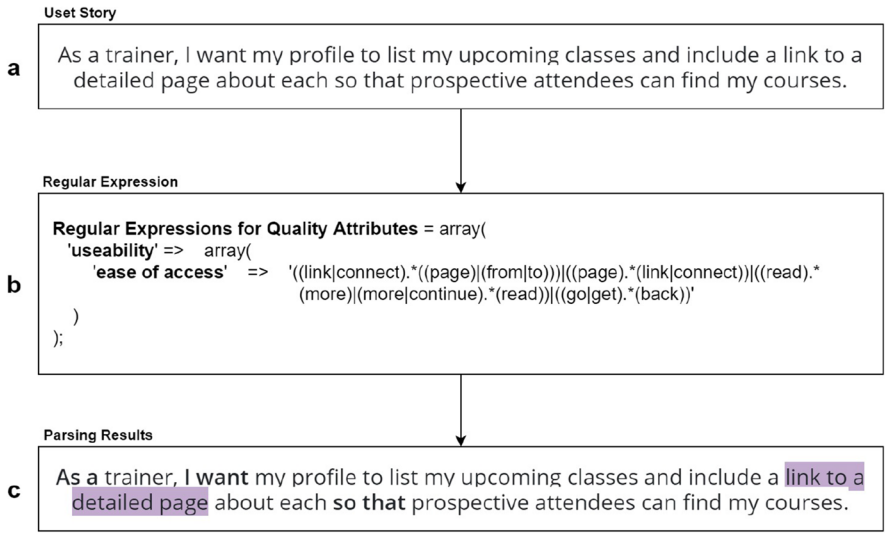

Fig. 5 Working illustration of QAExtractor

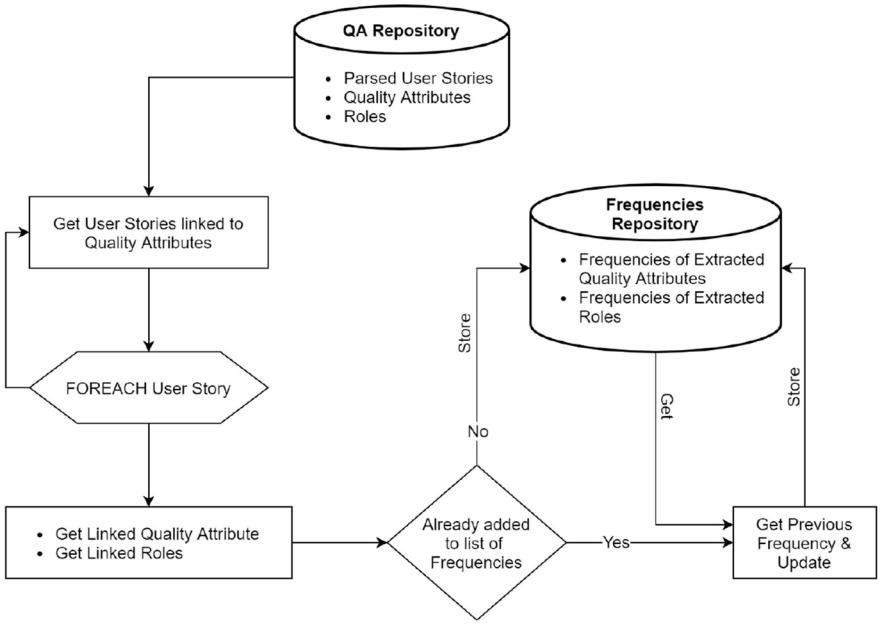

Fig. 6 Flow chart of frequency finder

roles, and finally save them into the frequencies repository. For the first time, the frequency of quality attribute and role would be set 1 and after that their frequencies would be updated to the number of appearances in the user stories. On the base of quality attribute's frequency, they are prioritized.

Second prioritization mechanism is based on the aggregation of quality attribute's frequencies with the role's frequency. Notice that only the frequency of a quality attribute is not enough to find its importance as the role also influences the quality level. So, a more dominant role in all user stories will have significant impact on the overall quality of software. The working flow is provided in Fig. 7. Firstly, we need to calculate the frequencies of each role. The frequency of role is the number of user stories that are attached to one role. Next, it is required to identify all stakeholders and maintaining their frequencies just as we proposed for the quality attributes frequencies (Fig. 7). Consider following requirement as an ex-ample:

"As a Broker user, I want to Upload and Validate the error message to have accurate text."

First, we extracted the quality attribute from this user story. Let's suppose it is "Abc", and its overall frequency in all user stories is "15". Then, we extracted Role of this user story, that is "Broker User" in this case. Then, we identified the overall frequency of this Role throughout the user stories. Let's suppose the fre-quency of "Broker User" is "9" . The fre-quency of Role means how many re-quirements are associated with this Role. Now, we have both frequencies, we ag-gregated them using following formula.

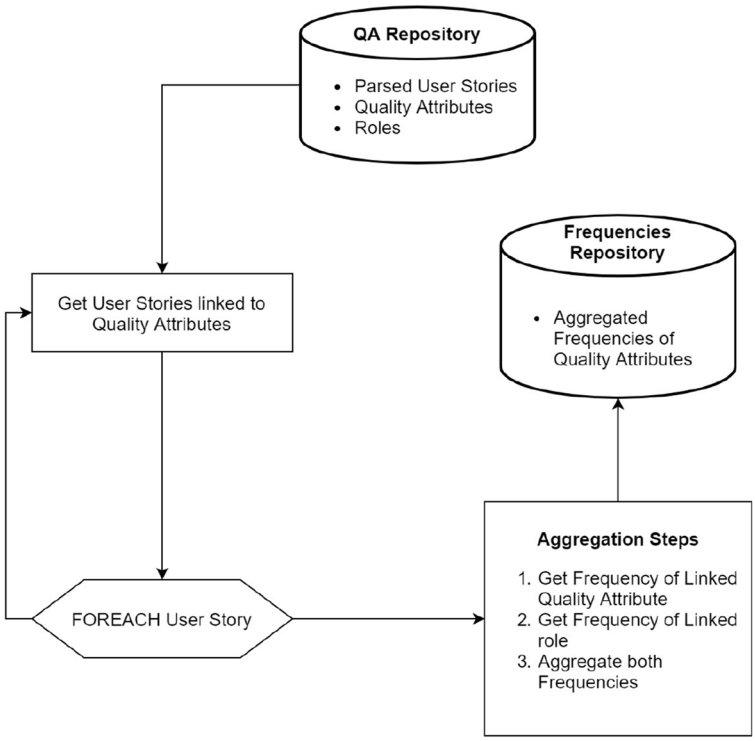

Fig. 7 Flow chart of quality attributes prioritization by aggregation mechanism

$$QA_{AV} = F_{QA} \times F_R$$

In above formula, QA_{AV} stands form Quality Attribute's Aggregated Value, F_{QA} stands for Frequency of Quality Attribute, and F_R stands for the Frequency of Role.

Third prioritization technique is based on the criticality of any quality attribute. The underlying concept of the criticality factor of quality attribute is adopted from the reality. Mostly, the QA manager or respective personnel defines that which quality attribute is critical to what extent. For this purpose, they ranks the quality attributes by assigning them a value from the range of 0-10 (or any other defined range). We exactly followed it and allowed the respective personnel to assign CFV for each quality attribute during the prioritization process. Later on, we aggregated these values with the frequencies of quality attributes to prioritise them.

It's not only relying on the input from the user. It also considers the requirements while prioritising the quality attributes. When it takes input from user, it also gets the intensity of that Quality At-tribute from the requirements and aggregates both values. So, if user will define right ranking for the quality attributes, most probably the system will rank the quality attributes accordingly. But, in case, if user makes a mistake while defining his ranking, the system's aggregation will reduce the impact of that false decision.

Rest of the mechanism and formula is pretty much similar as described above in Second Mechanism, just Value of Role is replaced by the CFV for each quality attribute. The CFV value is obtained from the user manually. Hence, the formula will be as following:

$$QA_{AV} = F_{QA} \times CFV$$

4 Results and analysis

This section presents the obtained experimental results of QAExtractor & QAPrioritisor.

4.1 QAExtractor

The proposed framework is implemented and validated on an online server using PHP 5.0 as a server scripting language. We used two open-source online datasets to validate our framework named as DS1 [30] and DS2 [31]. First, we tested our framework with DS1, and collected the results. After that, we revised the extraction rules (i.e. regular expressions) to improve the extraction process and provided DS2 to the proposed framework. We found that the extraction of quality attributes has been significantly improved by refining the extraction rules. In the end, we generated a custom quality model on the base of extracted quality factors and quality attributes. The proposed system successfully extracted 18 quality attributes. next, we classify them into 6 categories according to the McCall's quality model (McCall et al. 1977) for batter visualisation and understandings. The reason behind choosing McCall's quality model is that its sub-categories are further linked to low-level quality attributes such as Correctness is further composed of Consistency and Completeness. This provides the quality overview at more granular level. These quality factors and quality attributes are given in the Table 2.

Before proceeding with the results, let's have a brief discussion of correct and false detection of quality attributes by our proposed framework:

1. Example of non-functional attribute that was correctly detected by our proposed frame-work: "As a Developer , I want to be able to log better, so that I can troubleshoot issues with particular submissions and functions." The industrial practitioner labeled the attribute "Recovery" against this requirement which was successfully detected by our framework. Our framework specifically targeted they focus key-words "troubleshoot and issues" against which it detected "Recovery".
2. Example of non-functional attribute that was falsely detected by our framework: "As a user, I want to get feedback when I enter an invalid zip code." Our framework detected "Correctness" for this requirement which is a false detection. The rea-son behind the false detection is that the combi-nation of the keywords "Enter, Invalid and Code" triggered it for "Correctness". This is pret-ty much due to the false understanding of the context because the prior focus of the require-ment is

Table 2 The extracted quality factors and quality attributes

Quality factor	Quality Attribute
Security	Authenticity
	Integrity
Useability	Ease of access
	Operability
	Synthesis
Reliability	Correctness
	Completeness
	Consistency
	Stability
Maintainability	Flexibility
	Modularity
	Modifiability
Efficiency	Efficiency
	Recovery
	Performance
Understandability	Understandability
	Consistency
	Modularity

on "to get feedback". However, our framework is not doing this all the time. This is one of the few times when it failed.

From the testing of the proposed system, we observed that one user story can have more than one quality factors or quality attributes. Similarly, one quality factor or quality attribute can occur in more than one user stories. We employed quality assurance experts from the industry to label the user stories to benchmark the effectiveness of our proposed framework. On the basis of the labelled user stories, we formed a confusion matrix by calculating true positives, true negatives, false positives, and false negatives. The developed confusion matrix for DS1 is represented in Table 3.

Next, we quantify the effectiveness of the QAExtractor Framework, for DS1, by calculating precision, recall, and f-measure. The results of the employed performance metrics are given in Table 4. Notice that the obtained results are compared with a SpaCy Classifier (Gilson et al. 2019) and found that the proposed technique outperformed the benchmark technique in terms of accuracy, recall and f-measure.

Table 3 The confusion matrix formed by experimental results of DS1

n = 407	Actually positive	Actually negative
Predicted positive	True positives (297)	False positives (76)
Predicted negative	False negatives (26)	True negatives (8)

Table 4 Comparison of QAExtractor (for DS1) with spaCy classifier

Technique	Accuracy	Precision	Recall	F-Measure
SpaCy classifier (Gilson et al. 2019)	–	0.74	0.42	0.53
QAExtractor (DS1)	0.75	0.79	0.92	0.85

After experimentation with DS1, we revised and improved the extraction rules and again performed the experimentation for DS2. The confusion matrix for DS2 is given in Table 5.

The performance comparison of QAExtractor (for DS1 and DS2) with SpaCy Classifier is given in the Table 6.

From Table 6, it is obvious that QAExtractor outperforms in terms of the employed performance criteria. However, a significant difference is observed in the results of DS1 and DS2 that depicts the improvements in quality attributes extraction rules. The performance of QAExtractor is considerably increased in terms of Precision, and Recall.

Based on the results (as reported in Tables 3, 4, 5, and 6), it can be clearly observed that the proposed framework extracted a considerable amount of quality attributes successfully, both from the DS1 and DS2. First, we implemented the proposed technique on DS1. Among 1042 predictions, 799 predictions are correct that supports the accuracy of QAExtractor upto 75%. We observed that the reason behind the false positives is the comprehensive or complex user stories. For example, consider following user story:

"As a plan review staff member, I want to track and update the completion of required plan reviews, so that I can ensure all plan review tasks are completed, results have been sent to the applicant, and any downstream steps are initiated such has a final review or payment for permit issuance."

Surely, the above-mentioned complex user stories can cause ambiguity in extracting quality attributes. By considering this, we improved the proposed framework by refining the regular expressions. We mainly targeted the complex user stories and

Table 5 The confusion matrix formed through experimental results for DS2

n = 1042	Actually positive	Actually negative
Predicted positive	True positives (765)	False positives (112)
Predicted negative	False negatives (131)	True negatives (34)

Table 6 Comparison of QAExtractor (for DS1 and DS2) with existing fully automatic techniques

Technique	Accuracy	Precision	Recall	F-Measure
SpaCy classifier (Gilson et al. 2019)	–	0.74	0.42	0.53
Socio technical system (Knauss and Ott (2014))	–	0.58	0.62	0.52
AUR-BoW with Bagging (Lu and Liang 2017)	–	0.71	0.72	0.71
QAExtractor (DS1)	0.75	0.79	0.92	0.85
QAExtractor (DS2)	0.77	0.87	0.85	0.86

defined more rules to make the NLP-Parser more efficient to understand the complex nature of these user stories. After refining the propsoed framework, we implemented it on DS2 and observed slight improvement in terms of accuracy. This strengthened the cause of complex user stories and it could be concluded more confidently that the complex nature of user stories is a big hurdle in extracting quality attributes. Though the refinement slightly increased accuracy from 0.75 to 0.77, the precision is significantly increased from 0.79 to 0.87. This also indicates that there is strong need to specifically target the complex user stories to enhance the performance.

4.2 QAPrioritiser

After successfully extracting the quality attributes from the user stories, QAPrioritiser effectively prioritized the extracted quality attributes. We performed three prioritization mechanisms on the extracted quality attributes: (i) by frequency of quality attributes, (ii) by aggregating the frequency of quality attributes with the impact of roles on them, and (iii) by assigning the CFV to the quality attributes. The results of QAPrioritiser by the frequency of quality attributes (for DS2) is given in the Table 7. The quality attribute with maximum frequency is placed at the top, and the quality attribute with minimum frequency is placed at the bottom.

From Table 7, it can be clearly observed that the quality attribute "Correctness" is ranked at the top of all extracted quality attributes as it occurred maximum times in the user stories. In contrast, "Modifiability" and "Modularity" are placed at the bottom of the Table 7 as they occurred minimum times. After prioritising quality attributes by their frequency, QAPrioritiser prioritized them by aggregating the impact of roles on quality attributes. The results of the QAPrioritiser by the aggregating impact of roles (for DS2) is given in Table 8.

Table 7 Frequency of extracted quality attributes for DS2

Quality Attribute	Frequency	Percentage
Correctness	180	20.52
Operability	167	19.04
Ease of access	134	15.28
Consistency	99	11.29
Integrity	69	7.87
Flexibility	56	6.39
Synthesis	59	4.45
Completeness	36	4.1
Understandability	25	2.85
Authenticity	25	2.85
Efficiency	19	2.17
Recovery	10	1.14
Performance	8	0.91
Modifiability	5	5.57
Modularity	5	5.57

Table 8 Prioritization of quality attributes by aggregating roles, for DS2

Quality attribute	Aggregated frequency	Percentage
Operability	3675	30.6
Correctness	3576	29.77
Ease of Access	2190	18.24
Consistency	1088	9.06
Integrity	538	4.48
Flexibility	430	3.58
Completeness	164	1.37
Synthesis	147	1.23
Authenticity	75	0.63
Understandability	55	0.46
Efficiency	44	0.37
Recovery	12	0.1
Performance	8	0.07
Modularity	3	0.03
Modifiability	2	0.02

Table 8 shows the prioritization of extracted quality attributes with the help of aggregating impact of roles on quality attributes. From Tables 5 and 8, it can be observed that the prioritization results are slightly different. By aggregation with roles impact, "Operability" moved to the top. "Completeness" also moved one step upward and pushed "Synthesis" down. Similarly, "Authenticity" moved one step upward by pushing "Understandability" down. It is clear that the roles impact the prioritization of quality attributes.

Third, we prioritized quality attributes by assigning CFV to each quality attribute. For the testing, we assigned test CFV to each quality attribute. The range of CFV for each quality attribute is 1-10. The CFV 1 means that quality attribute is least important, and value 10 means that quality attribute is most important amongst others. The assigned test values are shown in Table 9.

The prioritization of quality attributes by their CFV values is shown in Table 10. It can be observed that the "Correctness" is moved to the top, where "Operability" is pushed one step down. "Flexibility" is also moved one step upward by pushing "Integrity" down. Moreover, "Efficiency" and "Performance" are also moved one step upward, by pushing "Understandability" and "Recovery" one step down, respectively. The difference in prioritization can also be observed as compared to Table 7.

5 Threats to validity

As indicated by the obtained results, we have successfully extracted software quality attributes and prioritized them. However, current research has certain limitations and threats to validity that have been clearly specified in the constituent sections. First, we review the external threats to validity to define the generalisation of the

Table 9 The CFV assigned to each quality attributes for DS2

Quality attribute	Assigned CFV
Operability	6
Correctness	8
Ease of access	10
Consistency	6
Integrity	4
Flexibility	6
Completeness	6
Synthesis	4
Authenticity	6
Understandability	2
Efficiency	8
Recovery	2
Performance	8
Modularity	4
Modifiability	8

Table 10 Prioritization of quality attributes by CFV for DS2

Quality attribute	Aggregated frequency	Percentage
Correctness	286	33.09
Operability	220	25.51
Ease of access	219	25.34
Consistency	65	7.55
Flexibility	25	2.99
Integrity	21	2.49
Completeness	9	1.14
Synthesis	5	0.68
Authenticity	4	0.52
Efficiency	3	0.41
Understandability	1.11	0.13
Performance	0.7	0.08
Recovery	0.25	0.03
Modularity	0.16	0.02
Modifiability	0.13	0.02

obtained results. Then, we present the internal threats to validity to be confident in cause-and-effect relationship to the targeted research. Finally, we discuss the construct and the conclusion validity.

5.1 External validity

Threats to external validity are the limitations to generalise the findings and results. In current research work, a threat to external validity is a limited numbers of datasets

used for the validation. So, we cannot conclude that the results will be the same for every type of dataset. The quality managers should be very careful while utilising this framework, especially for the health and safety-critical systems where a mistake can lead to a huge payoff. Furthermore, the proposed framework is only capable of identifying the 18 quality attributes shown in Table 1. It will not be able to identify any other quality attributes. Hence, we are unable to guarantee the identification of all software quality attributes with all types of the projects. For this purpose, we might need to explore other quality attributes and validate them with different types of datasets.

5.2 Internal validity

In this research context, the internal threat is the chosen dataset for the validation of the proposed framework. We utilised an open source dataset that was used by Gilson et al. (2019) for the validation of their proposed technique. However, we didn't verified and validated that how this dataset was collected by the authors. There is a possibility that the dataset might be wrong, incomplete, or having tempered records. It is also possible that the whole dataset is synthetic or artificial. Moreover, instead of labelling the dataset by ourselves, we hired quality assurance experts from the industry to perform this task. The experts labelled the quality attributes according to their understanding, knowledge, and experience. But, this labelling could cause biases as it was labelled by the experts from one software organisation. The view for the quality attributes can change organisation to organisation, even it could vary person to person.

5.3 Construct validity

Since there is very limited research in the domain of software quality attributes extraction, the only measurement or evaluation criteria found in the research are precision, recall, and f-measure. However, these evaluation criteria are widely used for the validation of any classification technique. The quality attributes extraction is also follows the basic principles of classification. So, these could be the most appropriate parameters to measure the proposed framework. However, the further research in the domain of quality attributes extraction might find some other parameter that could be more appropriate. Moreover, to the best of our knowledge, the prioritization of quality attributes has never been addressed in the past, so there doesn't exist any measurement criteria to evaluate its performance. To measure the prioritization of quality attributes is also based on the personal overview of experts and other parameters. So, in the future, we will be looking for a method to have a suitable guess of expert's overview about the quality attributes.

5.4 Conclusion validity

The conclusion validity is the degree of confidence about the relationship of the data and conclusion. In this research work, first threat to the conclusion validity is

the dataset and its labelling. We used two dataset: DS1 [30] is the sample dataset and DS2 [31] is an open source dataset used by Gilson et al. (2019). However, we labelled data through the employed quality assurance experts becauseGilson et al. (2019) didn't share their labelled data. So, in order to compare the obtained results with (Gilson et al. 2019), it is required to utilise a common labelled data. Second threat to the conclusion validity is the validation of QAPrioritiser. As far as the validation of prioritization of quality attributes is concerned, the only viable validation mechanism is through Expert's Opinion. This is mainly due to the fact that the prioritization of the quality attributes depends upon the need and interest of the customer, as well as the developers. A mechanism of continuous feedback could be adopted to set a benchmark inside an organization. Besides that, there couldn't be any hard and fast rule to set or define a benchmark for the validation of subjective analysis.

6 Conclusion and future directions

The research explicitly focuses on the identification and prioritization of quality attributes at early stages of SDL in ASD context. In this work, we proposed an NLP-based Quality Attributes Extraction and Prioritization Framework for automatic extraction and prioritization of quality attributes from user stories. The proposed framework successfully identified and extracted 18 quality attributes and prioritized them from the considered case studies. The obtained results showed that the proposed framework attained considerable effectiveness compared to the existing technique, in terms of precision, recall, and f-measure. The automation for quality attributes extraction and prioritization significantly reduced the time complexity of the manual process and minimised the need of experts. Through the proposed framework, a non-technical person could have a good overview of a software's quality. In future, we plan on refining the regular expressions to enhance the effectiveness of QAExtractor. Moreover, we also plan to target other quality attributes and design regular expressions for them. Furthermore, there is also need to identify some other prioritization criteria for QAPrioritiser to effectively rank the extracted quality attributes.

Acknowledgements The authors would like to acknowledge the feedback and suggestions of members of Software Reliability Engineering Group (SREG) that helps in improving the initial version of the manuscript.

References

Abad, Z.S.H., Karras, O., Ghazi, P., Glinz, M., Ruhe, G., Schneider, K.: What works better? a study of classifying requirements. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pages 496–501. IEEE, (2017)

Al Omran, F.N.A., Treude, C.: Choosing an nlp library for analyzing software documentation: a systematic literature review and a series of experiments. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pages 187–197. IEEE, (2017)

Al Imran, M.A., Lee, S.P., Ahsan, M.M.: Measuring impact factors to achieve conflict-free set of quality attributes. In: 2017 IEEE 8th Control and System Graduate Research Colloquium (ICSGRC), pages 174–178. IEEE, (2017)

Aljallabi, B.M., Mansour, A.: Enhancement approach for non-functional requirements analysis in agile environment. In: 2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE), pages 428–433. IEEE, (2015)

Arvanitou, Elvira Maria, Ampatzoglou, Apostolos, Chatzigeorgiou, Alexander, Galster, Matthias, Avgeriou, Paris: A mapping study on design-time quality attributes and metrics. J. Syst. Softw. **127**, 52–77 (2017)

Bellomo, S., Nord, R., Ozkaya, I.: Elaboration on an integrated architecture and requirement practice: prototyping with quality attribute focus. In: 2013 2nd International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks), pages 8–13. IEEE, (2013)

Chung, Lawrence, Nixon, Brian A., Yu, Eric, Mylopoulos, John: Non-Functional Requirements in Software Engineering, vol. 5. Springer, Berlin (2012)

Chung, L., Prado Leite, J.C.S.D.: On non-functional requirements in software engineering. In: Conceptual modeling: Foundations and applications, 363–379. Springer, Berlin (2009)

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)

Domah, D.: The NERV Methodology Non-Functional Requirements Elicitation Reasoning and Validation in Agile Processes PhD thesis. Nova Southeastern University, Fort Lauderdale (2013)

Etxeberria, L., Mendieta, G.S., Belategi, L.: Modelling Variation in Quality Attributes. VaMoS. **7**, 51–59 (2007)

Farid, W.M.: The normap methodology: Lightweight engineering of non-functional requirements for agile processes. In: 2012 19th Asia-Pacific Software Engineering Conference, volume 1, pages 322–325. IEEE, (2012)

Farid, W.M., Mitropoulos, F.J.: Normatic: A visual tool for modeling non-functional requirements in agile processes. In: 2012 Proceedings of IEEE Southeastcon, pages 1–8. IEEE, (2012)

Ferrari, Alessio, Dell'Orletta, Felice, Esuli, Andrea, Gervasi, Vincenzo, Gnesi, Stefania: Natural language requirements processing: a 4d vision. IEEE Ann. Hist. Comput. **34**(06), 28–35 (2017)

Gilson, F., Galster, M., Georis, F.: Extracting quality attributes from user stories for early architecture decision making. In: 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), pages 129–136. IEEE, (2019)

Hneif, M., Lee, S.P.: Using guidelines to improve quality in software nonfunctional attributes. IEEE Softw. **28**(6), 72–77 (2010)

Jain, P., Sharma, A., Ahuja, L.: Ism based identification of quality attributes for agile development. In: 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), pages 615–619. IEEE, (2016)

Jawad, A.N.A., Bashir, H.: Hierarchical structuring of organizational performance using interpretive structural modeling. In: 2015 International Conference on Industrial Engineering and Operations Management (IEOM), pages 1–7. IEEE, (2015)

Jeon, S., Han, M., Lee, E., Lee, K.: Quality attribute driven agile development. In: 2011 Ninth International Conference on Software Engineering Research, Management and Applications, pages 203–210. IEEE, (2011)

Kassab, M.: The changing landscape of requirements engineering practices over the past decade. In: 2015 IEEE fifth international workshop on empirical requirements engineering (EmpiRE), pages 1–8. IEEE, (2015)

Knauss, E., Ott, D.: (semi-) automatic categorization of natural language requirements. In: International Working Conference on Requirements Engineering: Foundation for Software Quality, pages 39–54. Springer, (2014)

Kurtanović, Z., Maalej, W.: Automatically classifying functional and non-functional requirements using supervised machine learning. In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pages 490–495. Ieee, (2017)

Lu, M., Liang, P.: Automatic classification of non-functional requirements from augmented app user reviews. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, pages 344–353, (2017)

Maiti, R.R., Mitropoulos, F.J.: Capturing, eliciting, and prioritizing (cep) nfrs in agile software engineering. In: SoutheastCon 2017, pages 1–7. IEEE, (2017)

McCall, J.A., Richards, P.K., Walters, G.F.: Factors in software quality. volume i. concepts and definitions of software quality. Technical report, GENERAL ELECTRIC CO SUNNYVALE CA, (1977)
Mendeley data - requirements data sets (user stories). https://data.mendeley.com/datasets/7zbk8zsd8y/1. (Accessed on 11/22/2020)
Moreira, A., Araújo, J., Brito, I.: Crosscutting quality attributes for requirements engineering. In: Proceedings of the 14th international conference on Software engineering and knowledge engineering, pages 167–174, (2002)
Petrov, S., Das, D., McDonald, R.: A universal part-of-speech tagset. arXiv preprint arXiv:1104.2086, (2011)
Rehman, Bisma, Alam, Khubaib Amjad, Ko, Kwang Man: Automated classification of mobile app reviews considering user's quality concerns. In: Sajid Anwar and Abdul Rauf, editors, Proceedings of the First International Workshop on Intelligent Software Automation, pages 29–35, Singapore, (2021). Springer Singapore
Schön, Eva-Maria., Thomaschewski, Jörg., Escalona, María José.: Agile requirements engineering: a systematic literature review. Comput. Stand. Interfaces **49**, 79–91 (2017)
User stories and user story examples by mike cohn. https://www.mountaingoatsoftware.com/agile/user-stories. (Accessed on 09/23/2020)
Yin, W., Kann, K., Yu, M., Schütze, H.: Comparative study of cnn and rnn for natural language processing. arXiv preprint arXiv:1702.01923, (2017)

Authors and Affiliations

Mohsin Ahmed[1] · Saif Ur Rehman Khan[1] · Khubaib Amjad Alam[2]

Mohsin Ahmed
its.mohsin.ahmed@gmail.com

Saif Ur Rehman Khan
saif_rehman@comsats.edu.pk

[1] Department of Computer Science, COMSATS University Islamabad, Islamabad, Pakistan

[2] Software Engineering Department, FAST School of Computing, National University of Computer and Emerging Sciences (FAST-NUCES), Islamabad, Pakistan