An Engineering Approach to Integrate Non-Functional Requirements (NFR) to Achieve High Quality Software Process

Muhammad Awais Gondal[1], Nauman A. Qureshi[2], Hamid Mukhtar[3] and Hafiz Farooq Ahmed[2]
[1]National University of Modern Languages (NUML), Islamabad, 44000, Pakistan
[2]CCSIT, King Faisal University, AlAhsa 31982, K.S.A.
[3]National University of Sciences and Technology (NUST), Islamabad, 44000, Pakistan

Keywords: Requirements Engineering, Quality Assurance, Non-Functional Requirements, Software Quality.

Abstract: Software quality calls for an engineering approach to incorporate non-functional requirements as first-class citizens into software specification and later operationalized at the development time. Recent research argues to model high level goals capturing the intentions of the users as Non-functional requirements (NFRs) at the early stage of the requirements engineering. However, intertwining relevant NFRs into the specification at early stage increases the complexity to many folds. Therefore, a straightforward approach for capturing NFRs is not possible as product specific NFRs are usually domain dependent. In this paper, we propose a systematic approach to integrate NFRs into the specification and development artifacts to ensure high quality of the software system under development. Considering existing seminal approaches in the literature, we propose a textual template for specifying NFRs and provide systematic technique to integrate relevant NFRs during the software requirements specification phase. We demonstrate our approach using a healthcare-information-systems as a case study and report initial results.

1 INTRODUCTION

In Requirements Engineering (RE), the Software Requirements Specification (SRS) document is the key artefact specifying all requirements of the software system (Davis et al., 1993). All the RE phases contribute to engineer requirements which are documented in the System Requirement Specification (SRS) document making it the most important artefact in the whole software development process (Nuseibeh and Easterbook, 2000). Problematic and low-quality definitions of requirements subsequently lead to a low-quality software product (Berry, 1995) (Maiti and Mitropoulos, 2017). Functional requirements (FRs) of a software product must be fulfilled; and at the same time satisfaction of non-functional requirements (NFRs) can lead to success or failure of the software (Ameller et al., 2012). NFRs have, therefore, a direct link with the software quality. NFRs are treated as constraints over functional or domain requirements which require user's input from an early phase of the RE process. Moreover, NFRs must also reflect quality attributes in general. Most, if not all, quality attributes defined in quality models of McCall et al. (1977), Boehm et al. (1978), Dromey (1995), ISO 9126 (1991) and FURPS (Grady, 1992), argue that explicit consideration of NFRs shall be given by the software engineer to incorporate relevant NFRs into the domain in which the software will be put into practice.

1.1 A Brief History

Until recently, NFRs are not implemented in the same way as Functional Requirements (Rosa et al. 2000). Mylopoulos et al. (1992) were among the first ones to formalize a framework that would incorporate NFRs into the development process rather than evaluating them in the final product.

As identified by Rosa et al. (2000), some of the reasons why it is too difficult to consider NFRs into the software development process include:

1. NFRs are more complex to deal with as compared to FRs
2. They are usually very abstract and stated only informally
3. They are rarely supported by tools, methodologies or languages

Gondal, M., Qureshi, N., Mukhtar, H. and Ahmed, H.
An Engineering Approach to Integrate Non-Functional Requirements (NFR) to Achieve High Quality Software Process.
DOI: 10.5220/0009568503770384
In *Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS 2020) - Volume 2*, pages 377-384
ISBN: 978-989-758-423-7

4. It is quite difficult to ensure if a specific NFR is satisfied by the final product or not
5. Very often NFRs conflict with each other
6. Instead of being considered by application programmers, they are taken as concerns of the environment builders.

Glinz (2007) has divided the problems associated with NFRs into three categories: definition problems, classification problems, and representation problems (see Figure 1). According to Glinz (2007), the various definitions of NFRs have discrepancies and they fail to provide one clear picture. Secondly, available classifications tend to give different idea of NFRs in different application contexts. The problems with representation of NFRs are also a challenge due to their fuzzy nature (Glinz, 2007). The kind of requirement (FR or NFR) depends on the way how we define it; if we change the way of its representation, an NFR may look like a FR in an SRS document. Lastly, documenting NFRs is also indicated as a representational problem. It is not agreed that one should include them in requirements definition section of SRS document or not (Glinz, 2007). According to most of the templates provided in IEEE Recommended Practice for Software Requirements Specifications (IEEE Computer Society, 1998), FRs and NFRs should be stated separately in specification documents. NFRs may also be attached to use cases wherever possible (other than global NFRs) (Jacobson et al. 1999).

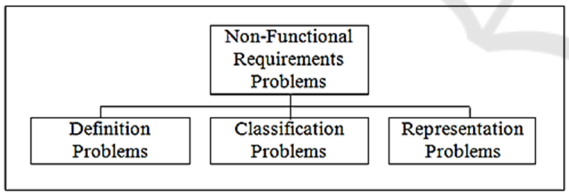

Figure 1: Non-functional Requirements Problems (Glinz, 2007).

NFRs can be represented in different ways depending on the rationale of their use and stage of the project. For instance, at early stage of the RE process, goal-oriented approaches provide a well-defined approach to model NFRs (Mylopoulos et al., 1992) (Chung et al., 2012). Moreover, architectural representation provides clear motivation to the designers to link architectural components with the NFR properties such that design can be justified.

Another very well-defined approach for representing NFRs is textual representation. While documenting requirements in SRS, we do require a specific template to incorporate NFR descriptions relevant to the FR and domain criterion. Natural language-based textual representations are the most widely used and easily adopted way of representing requirements (Davis, 2013) (Luisa et al., 2004) (Carvalho et al. 2015).

2 RELATED WORK

Various research works have been carried out that aim to incorporate NFRs in software development process. Some of these tend to be guidelines for this incorporation (Ormeno et al., 2013) (Martin et al., 2014) while others propose specific methodologies for achieving the mentioned goal, which are discussed in this section. As NFRs have their relationship with multiple stages and artefacts of software development process, the efforts in literature are also spread over these domains.

The work of Rosa et al. (2000) presents an approach for integrating NFRs in architecture through a formal model, but as identified by the authors, it is applicable only for dynamic software architectures. Paul et al. (2001) have tried to bridge the gap between architecture and NFRs by presenting a simple and direct mapping of requirements to architecture. The presented approach suggests that each software requirement explicitly or implicitly may contain information that is required for software architecture.

Along with architectural integration, design patterns focusing on NFRs can also be seen in literature. Liu and Yu (2004) and Carvalhaes et al. (2014) suggest that particular attributes of NFRs are satisfied by specific design patterns and using them in architecture design helps in achieving objectives associated with NFRs. Although less used and understood by software development community, using goal graphs for representing NFRs is also very common among research community (Bano et al., 2018). Goals are those objectives that the system must fulfil with cooperation of different associated actors (Liu and Yu, 2004).

The focus of this research paper is textual representations of NFRs that can be incorporated in SRS documents. Therefore, this section principally highlights such approaches that satisfy our focus. Volere Requirements Specification Template (Robertson and Robertson, 2006) is a widely used method for recording requirements in a structured way. Goals and requirements can be documented through Volere template with respect to their

rationale, associated stakeholder, priority and contextual details.

Different non-functional requirements like security, usability, maintainability etc. have specific templates presented in Volere documentation. Several studies like (Carvalhaes et al., 2014) (Porter et al., 2014) have used techniques inspired by Volere template, for their proposals.

A series of requirements templates provided by Duran et al. (1999) include non-functional requirements description template as well. Although, functional requirements templates provided in this template series have very concrete fields to be filled for describing functional requirements, it has been admitted that for non-functional requirements, the template is very general and does not have any specific field.

A problem with the usage of such templates is that they are useful only when a single person is responsible for managing them. However, in a project where many people are working simultaneously, this can lead to inconsistent, contradicting and omitted requirements, and a need for a complex requirements management tool (Nikula and Sajaniemi, 2002).

Moreira et al. (2002) have presented a model to identify quality attributes crosscutting functional requirements and to integrate them with functional requirements at an early stage of software development. The flow of our proposed approach is moderately related to this research work. However, unlike the authors, our approach is open and does not rely on specific modelling language like UML.

A number of research works propose NFRs templates that can be used for a specific NFR or quality attribute. An approach presented by He and Anton (2003) deals explicitly with privacy requirements modelling, while another similar approach based on use cases deals with efficiency requirements only (Dorr et al., 2003). The work by Chih-Wei, et al. (2006) proposes a model for dealing with performance requirements in agile development processes.

Apart from detailed tabular templates and models, several research works provide boilerplates (reusable sentences); a term first used by Hull and Jackson (2005) in the meanings of limited vocabulary sentences having specific placeholders to be completed in order to obtain semi-formal requirement sentences. Kopczyńska et al. (2014) have presented an elicitation methodology by the use of their Non-functional Requirements Templates (NoRTs), which focuses on using generic statements (having core and optional parts) that become defined NFRs after adding required information.

EARS presented by Mavin et al. (2009) also provides a simple boilerplate for requirement templates that can be used for non-functional requirements as well. Other such sentence templates can be seen in natural language-based boilerplates proposed by Ibrahim et al. (2014), and catalogues presented by Chung and Nixon (1995) and Cysneiros and Eric (2004).

Younas et al. (2019) use natural language processing techniques for identification of NFRs from requirements documents. The approach uses a language model and popular keywords for identification of NFRs. The empirical evidence shows that the automated semi-supervised approach reduces manual human effort in the identification of NFRs. This work suffers from the limitation of the lexicon or keywords as they are domain dependent. Also, the authors mention other limitations including the bias of the data labeller or the source of vocabulary for training the model.

With the emergence of Agile software development approaches, user stories (Cohn, 2004) have become the most popular way of expressing requirements focusing on the viewpoint of a role. However, the details required to capture the subtleties of NFR's cannot be captured by user stories by ordinary users.

The research works included in this section can be summarized as below:

a) The models to integrate NFRs are too generic to be practically adopted by software development units. They do not provide an implementation strategy for NFRs, thus leaving it up to the developer to document any such required strategy.

b) Most of the text-based tabular templates represent NFRs as independent elements of requirements process. The need of NFRs' relationship with specific functional requirements is not fulfilled by most of the efforts.

c) Boilerplates/sentence-oriented templates, on the other hand, lack the attribute of proper traceability and formal inclusion in SRS templates.

This paper specifically focuses on textual representation of NFRs that can be included in requirements documents with the aim of effective traceability and practical incorporation throughout the development process.

3 PROPOSED APPROACH

This section describes our proposed approach as represented in Figure 2. Table 1 presents the tabular text-based template used in our proposal. Following subsections provide details of each activity.

3.1 Identifying Functional Requirements

Software systems are based on a collection of functional and non-functional requirements. Several methods of elicitation of requirements are there that can be used to specify all requirements of the system under construction.

We start by specifying FRs and identifying actors associated with them as per use case approach. Some NFRs and quality attributes can be added at this stage according to the expertise of the developer and nature of the system under construction (Pressman, 2005), however, they are refined and identified in detail in the next step.

3.2 Identifying NFRs (Quality Factors)

As mentioned by Pressman (2005), specific implicit characteristics are expected from specific domains of software systems. Such common quality factors are usability as general for most of the applications, security for banking systems, and privacy for social networking applications. However, other quality factors must be analyzed that may impose constraints on the system or on a specific FR. Each quality factor can be further divided into specific quality attributes, for example, the usability factor can be divided in to attributes of task time, efficiency, and (user) error frequency.

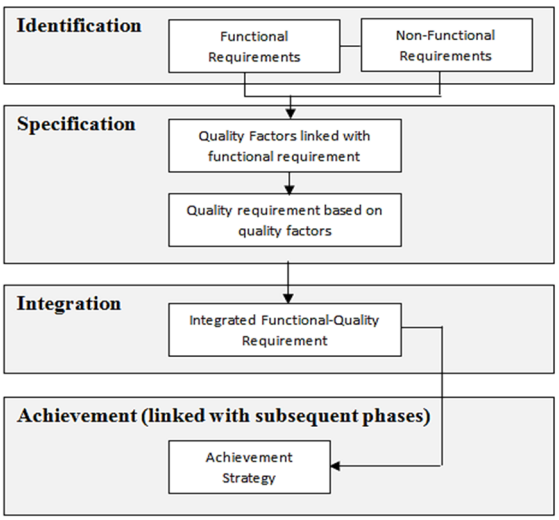

Figure 2: Proposed approach for integrating NFRs quality attributes.

We advocate creating a relationship of certain quality factors with FRs as fulfilling the quality factors at functional level will support NFRs at the global level. At this stage, we define the required quality factors for considered functional requirement.

3.3 Specifying Quality Requirements (with Respect to Quality Factors)

The specified quality factors and their attributes may have specific constraints and obligations associated with them. A *quality requirement* is defined at this stage that specifies the constraints or obligations linked with certain quality attributes. The quality requirement can be written in simple textual statements; however, we strongly support use of quantitative statements that can be measured as well at later stages. The specific quality attribute is highlighted in our proposed approach with '< >'. The specific constraint to the quality attribute is described with '{ }'. This is to guide developers to keep a record of quality factors and their achievement criteria (constraints) at the development stage.

3.4 Integrating Functional and Quality Requirements

Functional requirements and quality requirements should be integrated together in the form of Integrated Functional-Quality requirements. The basic functional requirement is enhanced to accommodate the conditions suggested by the quality requirement. The integrated requirements must include all constraints mentioned in quality requirement in relationship to the functional requirement in consideration (Glinz, 2007). The integrated functional-quality requirement is the final requirement description that will be used in subsequent phases of software development.

Inclusion of NFRs attributes in functional description will oblige the developers to consider them in all design decisions and phases. Since NFRs attributes are now a part of functional description, tracing them does not need a separate methodology; rather they are traced with functional requirements automatically.

Moreover, creation of test cases for specific functional requirements will also include NFRs attributes inevitably.

3.5 Formulating Achievement Strategy

At this earlier phase of software development, a specific strategy has to be specified for addressing the constraints imposed on functional description after integration of quality requirement.

A strategy can be specific design patterns, implementation techniques or simply conditions to be applied in code. At this stage, the strategy is for consideration purposes, and it is supposed to be explained in more detail during system design phase.

Table 1: Elements of Proposed NFRs Template.

NFR Template	
Functional Requirement	*(description)*
Quality Factors	*(description)*
Quality Factors Requirement	*(description)*
Integrated Functional-Quality Requirement	*(description)*
Achievement Approach Specification	*(description)*

4 CASE STUDY

For practical demonstration of the proposed approach and NFRs template, we developed a case study that consists of a simplified version of requirements from a Healthcare system practically deployed as mentioned in (Khan et al., 2017). Account of features required for the system is provided below.

"The Electronic Health Record (EHR) System for Obstetrics Specialty should provide the features of scheduling appointments for patients. Front desk staff should be able to collect basic information of the patients in order to register them. History of the patients should be gathered through the system and made available to the doctor where required. Alternatively, the doctor should be able to open the file (record) of a patient which is termed as Chart/Patient Visit file. The doctor can view lab reports/results of a specific patient. The system should allow the doctor to electronically prescribe medications. It should also allow the doctor to suggest lab tests for a specific patient".

A. Identifying Functional Requirements:

As the description is for an already developed system, here it is only presented in the form of Use Case diagram as shown in Figure 3. For presenting the process of creating our proposed template, we will utilize the use case of 'Access History'.

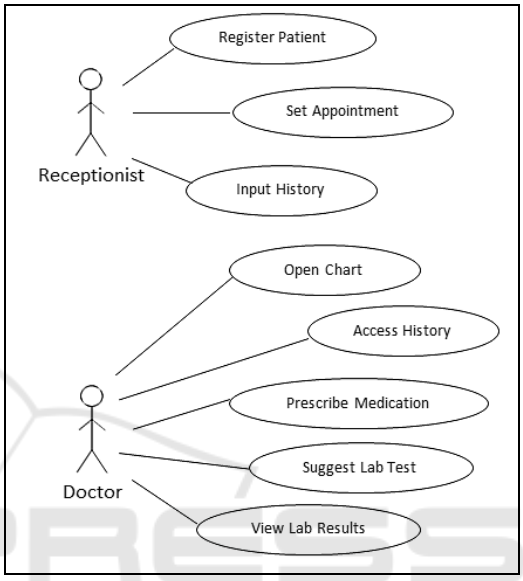

Figure 3: Use case diagram for case study.

B. Identifying NFRs (Quality Factors):

As mentioned in the previous section, at this stage we need to identify non-functional requirements (NFRs) as global properties of the system that provide certain constraints on functional requirements. Each functional requirement may have to fulfill particular constraints due to certain quality factors associated with the functional requirement. Here in this case the factor associated with the functional requirement is usability and it can be described in terms of its attributes that are task time and reduction of errors.

C. Specifying Quality Requirements (with Respect to Quality Factors):

In this stage certain constraints and obligations linked to the specified quality attributes are identified for a functional requirement. It provides a quality requirement with respect to the selected quality attributes. Table 2 shows identified quality attributes for a FR and its related quality requirement.

D. Integrating Functional and Quality Requirements:

The quality factor requirement specified in the previous stage can be integrated with the basic functional requirement to obtain functional-quality requirement. Table 3 provides the integration of both these requirements into one concrete requirement.

Table 2: Specification of Quality Requirement.

Functional Requirement	The system should present patient's history records to the user.
Quality Factors	Task time, (Reduction of) Errors
Quality Factors Requirement	<Average time> to interpret patient's history records should {be less than 5 minutes}. <Chances of error> while interpreting patient's history records should be {minimum i.e. up to 2 times}.

Table 3: Integration of Functional and Quality Requirements.

Functional Requirement	The system should present patient's history records to the user.
Quality Factors	Task time, (Reduction of) Errors
Quality Factors Requirement	<Average time> to interpret patient's history records should {be less than 5 minutes}. <Chances of error> while interpreting patient's history records should be {minimum i.e. up to 2 times}.
Integrated Functional-Quality Requirement	The system should present patient's history records that should be interpreted by the user within 5 minutes and with minimum error i.e. up to 2 times.

E. Formulating Achievement Strategy:

Different strategies for achieving NFRs and constraints can be followed depending on the nature of Integrated Functional-Quality Requirement and domain of the software project being constructed. The strategy for fulfilling the previously specified integrated functional-quality requirement has been mentioned in Table 4.

Our proposed NFRs template is completed with inclusion of achievement strategy.

Table 4: Achievement Approach Specification.

Functional Requirement	The system should present patient's history records to the user.
Quality Factors	Task time, (Reduction of) Errors
Quality Factors Requirement	<Average time> to interpret patient's history records should {be less than 5 minutes}. <Chances of error> while interpreting patient's history records should be {minimum i.e. up to 2 times}.
Integrated Functional-Quality Requirement	The system should present patient's history records that should be interpreted by the user within 5 minutes and with minimum error i.e. up to 2 times.
Achievement Approach Specification	Temporal layout and visuals Use temporal history view to reduce time. Use red colour to indicate alarming conditions in patient's history. Use yellow colour to indicate noteworthy situations in patient's history.

For clarification and a well-defined view, we are providing our proposed template for another use case i.e. 'Register Patient'. This template is presented in Tables 5.

Table 5: NFR Template for 'Register Patient'.

Functional Requirement	The system should present a form for collecting basic information of the user. The user should be able to fill in the required fields. The system should save all filled information.
Quality Factors	Task time, Completeness
Quality Factors Requirement	<Average time> to collect information should be {within 5 minutes} and it must be <Complete> with {no missing required data fields}.

Table 5: NFR Template for 'Register Patient' (cont.).

Integrated Functional-Quality Requirement	The system should present a form for collecting basic information. <Average time> for the user to fill in the required fields should be {within 5 minutes}. The system should only accept <Complete> information and {no missing required data fields} should be there. The system should save all filled information.
Achievement Approach Specification	Auto-complete, Warning messages Use auto-complete and pre-selected fields to reduce task time. Divide the information entry fields in categories of highly required, required and optional. Do not allow the user to leave any 'highly required' field blank. Show a colour-based warning on leaving 'required' fields blank.

5 CONCLUSIONS

In this paper, we presented an easy approach for integrating NFRs in the SRS documents in an intuitive way. We have provided details of the process of integration and practically shown our proposed template based on a case study.

The proposed approach is based on natural language textual representation, which is one of the most commonly used methods. It will ensure early consideration of NFRs in the software development process. Apart from NFR integration, we have also extended our proposed template to achievement methodology specification which will help in finding practical solutions for fulfilling constraints imposed by specific NFRs, at an early stage of software development.

In future, we aim to explore the relationship of multiple functional requirements in order to fulfil one specific non-functional requirement or constraint. For example, integrity requirements as per McCall's quality model are linked to authorized access to features of the system. Let's assume that in a system, only admins can access certain features. In such a scenario, these features might include multiple functional requirements which together will satisfy overall integrity requirements. Such an approach might require creating separate tables for collection of related functional requirements. Apart from it, some NFR constraints can be related to data and not to functional requirements.

REFERENCES

Ameller, D., Ayala, C., Cabot, J., & Franch, X. (2012). Non-functional requirements in architectural decision making. *IEEE software*, *30*(2), 61-67.

Bano, J., Reddy, L. S. S., & Khammari, H. (2018). A Significant Research Framework on Goal Oriented Requirement Engineering. *International Journal of Applied Engineering Research*, *13*(11), 9558-9564.

Berry, D. M. (1995). The importance of ignorance in requirements engineering. *Journal of Systems and Software*.

Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., & MacLeod, G. (1978). Merritt.: Characteristics of Software Quality.

Carvalhaes, M. F., da Rocha, A. F., Vieira, M. F., & Barbosa, T. M. D. A. (2014). Affective embedded systems: a requirement engineering aproach. *International Journal of Computer Trends and Technology (IJCTT)*, *8*, 70-75.

Carvalho, Gustavo, et al. "NAT2TEST tool: From natural language requirements to test cases based on CSP." Software Engineering and Formal Methods. Springer, Cham, 2015. 283-290.

Chung, L., & Nixon, B. A. (1995, April). Dealing with non-functional requirements: three experimental studies of a process-oriented approach. In *1995 17th International Conference on Software Engineering* (pp. 25-25). IEEE.

Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). *Non-functional requirements in software engineering* (Vol. 5). Springer Science & Business Media.

Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.

Cysneiros, L. M., & Yu, E. (2004). Non-functional requirements elicitation. In *Perspectives on software requirements* (pp. 115-138). Springer, Boston, MA.

Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., & Ta, A. (1993, May). Identifying and measuring quality in a software requirements specification. In *[1993] Proceedings First International Software Metrics Symposium* (pp. 141-152). IEEE.

Davis, A. (2013). *Just enough requirements management: where software development meets marketing*. Addison-Wesley.

Dörr, J., Kerkow, D., Von Knethen, A., & Paech, B. (2003, June). Eliciting efficiency requirements with use cases. In *Ninth international workshop on requirements engineering: foundation for software quality. In conjunction with CAiSE* (Vol. 3).

Dromey, R. G. (1995). A model for software product quality. *IEEE Transactions on software engineering*, *21*(2), 146-162.

Durán Toro, A., Bernárdez Jiménez, B., Ruiz Cortés, A., & Toro Bonilla, M. (1999). A requirements elicitation approach based in templates and patterns.

Glinz, M. (2007, October). On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)* (pp. 21-26). IEEE.

Grady, R. B. (1992). *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc.

Grunbacher, P., Egyed, A., & Medvidovic, N. (2001, August). Reconciling software requirements and architectures: The cbsp approach. In *Proceedings Fifth IEEE International Symposium on Requirements Engineering* (pp. 202-211). IEEE.

He, Q., & Antón, A. I. (2003, June). A framework for modeling privacy requirements in role engineering. In *Proc. of REFSQ* (Vol. 3, pp. 137-146).

Ho, C. W., Johnson, M. J., Williams, L., & Maximilien, E. M. (2006, July). On agile performance requirements specification and testing. In *AGILE 2006 (AGILE'06)* (pp. 6-pp). IEEE.

Hull, E., Jackson, K., & Dick, J. (2005). Advanced traceability. *Requirements Engineering*, 131-151.

Ibrahim, N., Kadir, W. M. W., & Deris, S. (2014, September). Documenting requirements specifications using natural language requirements boilerplates. In *2014 8th. Malaysian Software Engineering Conference (MySEC)* (pp. 19-24). IEEE.

IEEE Computer Society. Software Engineering Standards Committee, & IEEE-SA Standards Board. (1998). IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineers.

ISO, I. (1991). Information technology-software product evaluation-quality characteristics and guidelines for their use. *ISO/IEC IS*, *9126*.

Jacobson, Ivar et al. (1999). The unified software development process. Vol. 1. Reading: Addison-Wesley.

Khan, A., Mukhtar, H., Ahmad, H. F., Gondal, M. A., & Ilyas, Q. M. (2017, March). Improving Usability through Enhanced Visualization in Healthcare. In *2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS)* (pp. 39-44). IEEE.

Kopczyńska, S., & Nawrocki, J. (2014, August). Using non-functional requirements templates for elicitation: A case study. In *2014 IEEE 4th International Workshop on Requirements Patterns (RePa)* (pp. 47-54). IEEE.

Liu, L., & Yu, E. (2004). Designing information systems in social context: a goal and scenario modelling approach. *Information systems*, *29*(2), 187-203.

Luisa, M., Mariangela, F., & Pierluigi, N. I. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, *9*(1), 40-56.

Maiti, R. R., & Mitropoulos, F. J. (2017, March). Capturing, eliciting, and prioritizing (CEP) NFRs in agile software engineering. In *SoutheastCon 2017* (pp. 1-7). IEEE.

Martín, Y. S., Del Alamo, J. M., & Yelmo, J. C. (2014, August). Engineering privacy requirements valuable lessons from another realm. In *2014 IEEE 1st International Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE)* (pp. 19-24). IEEE.

Mavin, A., Wilkinson, P., Harwood, A., & Novak, M. (2009, August). Easy approach to requirements syntax (EARS). In *2009 17th IEEE International Requirements Engineering Conference* (pp. 317-322). IEEE.

McCall, J. A., Richards, P. K., & Walters, G. F. (1977). *Factors in software quality. volume i. concepts and definitions of software quality*. General Electric Co Sunnyvale, CA.

Moreira, A., Araújo, J., & Brito, I. (2002, July). Crosscutting quality attributes for requirements engineering. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering* (pp. 167-174).

Mylopoulos, J., Chung, L., & Nixon, B. (1992). Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on software engineering*, (6), 483-497.

Nikula, U., & Sajaniemi, J. (2002, September). Basyre: A lightweight combination of proven RE techniques. In *Proceedings of the International Workshop on Time Constrained Requirements Engineering*.

Nuseibeh, B., & Easterbrook, S. (2000, May). Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 35-46).

Ormeño, Y. I., Panach, J. I., Condori-Fern, N., & Pastor, Ó. (2013, May). Towards a proposal to capture usability requirements through guidelines. In *IEEE 7th International Conference on Research Challenges in Information Science (RCIS)* (pp. 1-12). IEEE.

Porter, C., Letier, E., & Sasse, M. A. (2014, August). Building a National E-Service using Sentire experience report on the use of Sentire: A volere-based requirements framework driven by calibrated personas and simulated user feedback. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)* (pp. 374-383). IEEE.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave macmillan.

Robertson, J., & Robertson, S. (2006). Volere: Requirements specification template (2006). *The Atlantic Systems Guild*, *47*.

Rosa, N. S., Justo, G. R., & Cunha, P. R. (2000, May). Incorporating non-functional requirements into software architectures. In *International Parallel and Distributed Processing Symposium* (pp. 1009-1018). Springer, Berlin, Heidelberg.

Younas, M., Wakil, K., Jawawi, D. N., Shah, M. A., & Mustafa, A (2019). An Automated Approach for Identification of Non-Functional Requirements using Word2Vec Model.