IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 3 April 2023, accepted 25 May 2023, date of publication 29 May 2023, date of current version 7 June 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3281195*

RESEARCH ARTICLE

# Proposed Framework to Manage Non-Functional Requirements in Agile

**EZELDIN SHERIF[1], WALEED HELMY[1], AND GALAL HASSAN GALAL-EDEEN[2]**
[1]Information Systems Department, Faculty of Computers and AI, Cairo University, Cairo 12613, Egypt
[2]Management Department, School of Business, The American University in Cairo, Cairo 11835, Egypt

Corresponding author: Ezeldin Sherif (Ezeldinsherif@gmail.com)

**ABSTRACT** Agile Software Development (ASD) is a type of iterated software development that strives to maximize productivity, effectiveness, and quick delivery through the minimization of documents and needless procedures within constrained timeframes. Agile software development has a number of advantages. There are still some difficulties. For instance, during the development lifecycle, non-functional requirements (NFRs) are disregarded and not given first-class artifacts. This results in several issues, including customer dissatisfaction and a great deal of rework, which impacts time and cost. In this paper, a proposed framework for handling non-functional requirements in Agile is explained. The framework supports the several primary activities of requirements engineering including requirements elicitation, analysis, documentation, and validation. In addition, the framework handles non-functional recommendations. Results of the suggested solution validation showed that it could address the problems with non-functional requirements in Agile.

**INDEX TERMS** Agile, non-functional requirements, scrum.

## I. INTRODUCTION

The term ''Agile'' denotes flexibility, adaptation, and agility [1]. Agile software development entails the division of a single large software project into numerous independently executable sub-projects. The development team begins by focusing on the most important features that users have chosen, other features chosen in successive iterations, and so on. Due to its advantages, such as increased team productivity, motivation, and software quality, Agile software development has grown in popularity over the past ten years [1]. However, there are still some challenges and issues in Agile Software Development [2]. One of the main challenges is not focusing on non-functional requirements [2], [3], [4], [5], [6], [7], [8], [9]. Our research problem in this challenge is focused on recommendation, elicitation, analysis, documentation, and validation of non-functional requirements in Agile. Therefore, this paper proposes a framework to address this challenge. This paper is organized as follows: Section one provides an introduction, Section two presents the research methodology, Section three presents the Agile Software Development Background, Section four presents a proposed framework, Section five presents validation and results, and finally, Section six outlines the conclusion of this paper.

The associate editor coordinating the review of this manuscript and approving it for publication was Aasia Khanum.

## II. RESEARCH METHODOLOGY

To analyze key works now available in the non-functional requirements in Agile, this research report carried out a critical literature review of this area. This research was conducted in four phases. The existing related articles were first screened using popular search engines such as IEEE Xplore Digital Library, Google Scholar, ACM, Springer, Science Direct, and Wiley Online Library. A keyword-based search on the terms was used throughout the screening process using different combinations of the following terms: ''non-functional requirement'', ''scrum'', Agile'', ''quality requirement'', ''requirements engineering in Agile'', ''conflict'', ''requirements elicitation'', and ''recommendation''. The second phase concentrated on selecting the important key works that would be examined and analyzed by filtering the collected papers. All pertinent papers were divided into categories based on their origin, whether they were theoretical or practical, and where they were published, whether in journals or conference proceedings. The third

phase involved developing an analysis of the filtered works to identify their advantages and disadvantages as regards Agile solutions for focusing on non-functional requirements. Finally, the result from the analytical study had been reached by the fourth phase.

## III. AGILE SOFTWARE DEVELOPMENT BACKGROUND

Agile Software Development let the development team first concentrates on the most important features that people have asked for. The team uses an incremental iterative development approach for all of the chosen capabilities, with each iteration producing a working system. The focus of the development team is on adapting quickly to changing requirements. The development team promptly modifies the plan in response to requests for changes to the requirements. Seventeen software developers got together in 2001 to talk about a quick and efficient development process [10]. They published a document called ''Manifesto for Agile Software Development''. The Agile manifesto contains twelve principles and four values which drew on their collective experience in software development and recognized the need to move away from rigid process models like Waterfall. The four values are [10]:

1) Individuals and interactions over processes and tools.
2) Working software over comprehensive documentation.
3) Customer collaboration over contract negotiation.
4) Responding to a change over following a plan.

The twelve principles are [10]:

1) The highest priority is to satisfy the customer through the early and continuous delivery of valuable software.
2) Welcome changing requirements, even late the development. Agile processes harness change for the customer's competitive advantage.
3) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for a shorter timescale.
4) Business people and developers must work together daily throughout the project.
5) Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
6) The most efficient and effective method by which to convey information to and within a development team is face-to-face conversation.
7) Working software is the primary measure of progress.
8) Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9) Paying continuous attention to technical excellence and good design enhances agility.
10) Simplicity -the art of maximizing the amount of work not done- is essential.
11) The best architectures, requirements, and designs emerge from selforganizing teams.
12) At regular intervals, the team should reflect on how to become more effective, then tune and adjust its behavior accordingly.

The above principles and values aim to accomplish two main objectives. The first objective is increasing public knowledge of Agile methodologies is the primary goal. The second objective is to help project teams decide if they are going to use an Agile process or not. This manifesto defines the philosophy behind Agile techniques, together with all of its guiding principles and values, and should ideally be included in all of the Agile methods' practices [11]. These principles fall short of taking into account the significance of the non-functional requirements (NFRs) [12]. As shown in Table 1, there are 16 different challenges in Agile requirements engineering discussed by 15 different research papers [2]. The challenge labeled with 'Y' means this challenge is mentioned in this given paper. The challenge that is labeled with '−' means that this challenge is not mentioned in the given paper. Neglecting non-functional requirements in Agile along with minimal documentation are the top two challenges in Agile requirements engineering [2], as these two challenges are stated in 14 out of the 15 research papers. Our focus is on the non-functional requirements in Agile.

There are related solutions that did focus on non-functional requirements in Agile software development. As shown in Table 2, it consists of 4 methodologies and 3 methods that focus on non-functional requirements in the Agile software development [2]. Table 3 consists of 3 tools, 2 frameworks, 1 metric, 1 study, 1 process, 1 guideline, and 1 new artifact that focuses on non-functional requirements in the Agile software development [2]. Each one of those solutions tackled the challenge of non-functional requirements in Agile from different perspectives. As shown in Tables 2 and 3, the keywords ''yes'' and ''no'' indicate whether or not the item applies to the given solution or research article, respectively. As a summary of both tables, there is no single solution that focuses on recommendation, elicitation, analysis, documentation, and validation of non-functional requirements in Agile.

## IV. MANoR: A PROPOSED FRAMEWORK

In any given software, there are two types of requirements: functional and non-functional requirements. This also applies to any software developed using an Agile software development approach. One of the foci in ASD is to deliver working functional software to the client as soon as possible. Therefore, the focus is on the functional requirements and Agile software development treats them as a first-class object. So, there is less focus on non-functional requirements. One of the problems of neglecting requirements, in general, is rework, where extra work is needed to fix problems in software due to poorly understood requirements [43]. The rework can cost from 40% to 50% of the total efforts of the software project [43]. It is found that a failure range of 60% or higher of software products when not considering non-functional requirements while development [9], [44], [45], [30], [46], [29], [36]. There are many different points to be considered while working on non-functional requirements, such as how are non-functional requirements going to be elicited, documented, and validated.

**TABLE 1.** Challenges in Agile requirements engineering adopted from [2].

| No | Challenge | [13] | [5] | [6] | [14] | [15] | [16] | [17] | [18] | [19] | [20] | [21] | [22] | [23] | [24] | [25] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Neglect of non-functional requirement | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | - | Y | Y | Y |
| 2 | Minimal documentation | Y | Y | - | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| 3 | Inadequate/inappropriate architecture | Y | - | Y | - | Y | - | - | - | Y | Y | Y | Y | Y | Y | Y |
| 4 | Prioritization of non-functional requirement | Y | - | - | - | - | - | - | Y | - | - | Y | - | - | Y | Y |
| 5 | Non-functional requirement infeasibility | - | - | Y | - | - | Y | - | - | - | - | - | - | - | - | - |
| 6 | Teams interaction | - | - | Y | - | - | - | - | - | - | - | - | Y | - | - | - |
| 7 | Inadequate non-functional requirement verification | - | - | Y | Y | - | - | - | - | - | - | - | Y | - | Y | - |
| 8 | Non-functional requirements identification | - | - | Y | - | - | Y | - | - | - | - | Y | - | - | - | - |
| 9 | Customer unavailability | - | - | - | Y | Y | - | - | Y | Y | Y | Y | Y | Y | Y | Y |
| 10 | Project budget and time estimation | - | - | - | - | Y | Y | - | Y | Y | Y | Y | Y | Y | Y | Y |
| 11 | Change of requirements | - | - | - | - | Y | Y | - | - | Y | Y | Y | - | - | - | Y |
| 12 | Problems related to prototyping | - | - | - | - | - | - | Y | - | - | - | - | - | - | - | - |
| 13 | Not understanding the big picture | - | - | - | - | - | - | Y | - | - | - | - | - | - | - | - |
| 14 | Lack of requirements traceability | - | - | - | - | - | - | - | Y | - | - | - | - | - | - | - |
| 15 | Contractual limitations | - | - | - | - | - | - | - | Y | Y | Y | - | - | - | - | Y |
| 16 | Customer inability and agreement | - | - | - | - | - | - | - | - | Y | Y | Y | - | - | - | - |

**TABLE 2.** Different methods/methodologies to handle non-functional requirements in Agile [2].

| Item/Paper | 1 NORMAP [26] | 2 NERV [27] | 3 MEDoV [28] | 4 CEP [29] | 5 Decision Tree [30] | 6 ACRUM [31] | 7 SENSoR [32] |
|---|---|---|---|---|---|---|---|
| **Type of solution** | Methodology | Methodology | Method | Methodology | Methodology | Method | Method |
| **Inform client meaning of NFR** | No | No | No | No | No | No | Yes |
| **Handles functional requirements** | Yes | Yes | No | Yes | No | Yes | No |
| **Elicit NFR** | No | Yes | Yes | Yes | No | No | Yes |
| **Analyze NFR** | No | Yes | Yes | No | No | Yes | Yes |
| **Link NFR with functional requirements** | Yes | Yes | No | No | No | Yes | No |
| **Model NFR** | Yes | No | Yes | No | No | No | No |
| **Recommend NFR** | No | No | No | No | No | No | No |
| **Plan (prioritized) NFR** | No | No | No | Yes | No | No | No |
| **Schedule NFR** | No | No | No | No | No | No | No |
| **Predict NFR** | No | No | No | No | Yes | No | No |
| **Document NFR** | Yes | Yes | Yes | Yes | No | Yes | Yes |
| **Validate NFR** | No | No | Yes | No | No | Yes | Yes |

**TABLE 3.** Other solutions to handle non-functional requirements in Agile [2].

| Item/Paper | 1 NORMATIC [33] | 2 NORPLAN [34] | 3 NORVIEW [35] | 4 CEPP [36] | 5 NFRec Tool [37] | 6 NFR recommendation System [38] | 7 AgileRE [39] | 8 Elicitation Guideline [40] | 9 Requirements elicitation [41] | 10 Modeling NFR [42] |
|---|---|---|---|---|---|---|---|---|---|---|
| **Type of solution** | Tool | Metric | Framework | Study | Tool | Process | Tool | Guidelines | Framework | New artifacts |
| **Inform client meaning of NFR** | No | No | No | No | No | No | No | No | No | No |
| **Handles functional requirements** | Yes | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes |
| **Elicit NFR** | No | No | No | Yes | Yes | No | No | Yes | Yes | No |
| **Analyze NFR** | No | No | No | No | No | No | No | No | No | No |
| **Link NFR with functional requirements** | No | No | No | No | Yes | No | Yes | Yes | No | Yes |
| **Model NFR** | Yes | No | No | No | No | No | No | No | No | Yes |
| **Recommend NFR** | No | No | No | No | Yes | Yes | No | No | No | No |
| **Plan (prioritized) NFR** | No | Yes | No | Yes | No | No | No | No | No | No |
| **Schedule NFR** | No | No | Yes | No | No | No | No | No | No | No |
| **Predict NFR** | No | No | No | Yes | No | No | No | No | No | No |
| **Document NFR** | Yes | Yes | Yes | Yes | Yes | No | Yes | Yes | Yes | No |
| **Validate NFR** | No | No | No | No | No | Yes | Yes | No | No | No |

Another point is whether there is a possibility to recommend certain non-functional requirements to the client. What if there is a conflict between non-functional requirements? This is where the proposed framework comes in. The proposed framework is called MANoR which stands for "Managing Agile Non-functional Requirements." The framework assists in resolving the above issues, regarding the non-functional requirements. The framework consists of different stages, and components; it is supported by a tool with the same name as the framework, as shown in Figure 1. The tool is a web application developed using ASP.NET MVC framework and the tool is available on[1] and credentials are available on.[2] The tool includes 14 different main features that consist of 48 different functions. The tool handles both functional requirements and non-functional requirements in Agile, more specifically, Scrum. Regarding the framework, there are two main stages and five main components. The stages are pre-analysis and post-analysis. The components are non-functional requirements recommendation, elicitation, analysis, documentation, and validation. The following are the main steps of the framework:

1) MANoR recommends specific non-functional requirements for the software being developed based on specific inputs. The recommendation will be based on two main items. The first item is historical data (previous projects). The second item is predefined data gathered from research and entered into the MANoR tool. Predefined data includes characteristics of the software such as: is the software being developed is

[1] http://manor-app.org/

[2] username: Manor_guest, password: Mg!AuNeOs2Rt

a web application, a desktop application, or a mobile application. In addition, data includes the business domain for which the software is being developed.
2) Non-functional requirements elicitation session occurs between the product owner/business analyst and key stakeholders from the client side in order to determine non-functional requirements. The objective of this session is to explain the meaning of the concept of non-functional requirements to the stakeholders. In addition, a list of recommended non-functional requirements can be provided by MANoR based on the characteristics of the software that will be developed.
3) The next step is to find conflicts in non-functional requirements. MANoR will assist in finding those conflicts.
4) Functional and non-functional requirements are documented in the product backlog by the product owner in MANoR. The documentation is written as user stories and MANoR provides different types of user stories (non-functional and functional).
5) Non-functional requirements will be validated by the client before starting the development.
6) Once step 5 is done, the normal process of Scrum will continue.

There are five main components of the framework: non-functional requirements recommendation, elicitation, analysis, documentation, and validation. Each of those components is explained further in the below sections.

### A. MANoR: STAGE I

In this section, we discuss the first two components of the framework. The first component is the requirements recommendation which is discussed in section I. The second component is requirements elicitation which is discussed in section II.

#### 1) REQUIREMENTS RECOMMENDATION

Research has shown historical data to be useful in determining a future event based on past data [30]. Therefore, the proposed solution is to recommend non-functional requirements based on historical data in ''MANoR''. In addition, a recommendation can be made based on data found in table 4 [47] and table 5 [47], [48].

Eight distinct application domains were taken into consideration, as indicated in Table 4, including transportation, banking and finance, telecommunication services, education, medical/health care, energy resources, insurance, government, and military [47]. In addition, the table shows that usability and performance are considered in seven domains, security is considered in six domains, and reliability is considered in four domains.

As shown in table 5 [47], [48], there are five different types of systems: real-time systems, process-controlled systems, safety-critical systems, information systems, and web systems [47], [48]. In addition, it shows that security, performance, and usability are considered in all five types of systems, and reliability is considered in four types of systems.

Based on the above information, our proposed solution recommends non-functional based on the following:

1) An application domain (predefined data from table 4)
Based on table 4, the data is entered in the ''MANoR'' which helps users with relevant non-functional requirements based on various application domains. This becomes extremely helpful when there is no historical data available.
2) Type of systems (predefined data from table 5)
The relevant data from table 5 is entered into ''MANoR'' which assists the users with relevant types of non-functional requirements based on different types of systems. This data includes relevant non-functional requirements based on the type of system. For example, communicativeness, dependability, reliability, performance, security, usability, integrity, and safety are relevant non-functional requirements for safety-critical systems.
3) Historical data
This option allows the tool to use the previous projects that were entered in ''MANoR''. As more projects are entered into the tool, it can provide better recommendations for upcoming projects. However, for the first few projects being entered into the tool, recommendations can be made through the application domain or the type of systems. Using the historical data inside the MANoR tool, it recommends non-functional requirement(s) based on a specific domain and a specific type of software being developed. The recommendation is based on how many non-functional requirements the end user wants to see. For example, if the client wants to know the top two non-functional that were used in previous projects with the same type of software and the same type of business domain, then the tool provides the results by showing two non-functional requirements based on the historical data. The end user can request to view the top one or two or three or four or five or even ten non-functional requirements that were used previously in the same type of software and the same type of business domain. Another example: let's assume that we have five different projects inside the tool that are involved in the academic domain and all of them are web applications, and currently we will work on the sixth project of the same type of software and the same type of domain. In addition, all five projects had one non-functional in common which is performance and four out of the five projects had security as a non-functional requirement. Then the end user requested to view the top two non-functional requirements that were mostly used in this domain and type of system. Therefore, the tool will display performance with a percentage of 100% and security with a percentage of 80%. The percentage is calculated within the tool by knowing that five projects out of a total of five projects did use performance as a non-functional requirement and also four projects out of five projects did use security as a non-functional requirement.

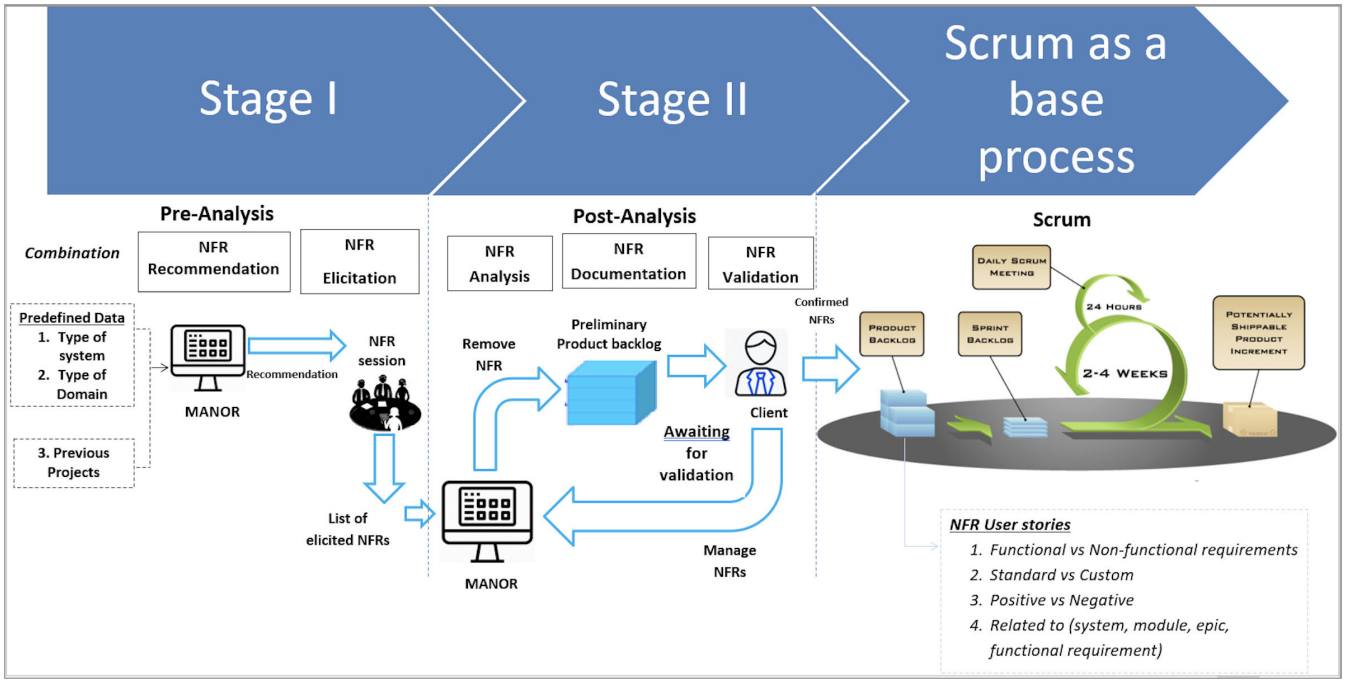



**FIGURE 1.** MANoR framework.

**TABLE 4.** Application domain & its relevant non-functional requirements, adopted from [47].

| ID | Domain | Recommended NFRs |
|---|---|---|
| 1 | Banking and Finance | Interoperability,performance,reliability,scalability,security,usability |
| 2 | Education | Interoperability,performance,reliability,scalability,security,usability |
| 3 | Energy Resources | Availability,performance,reliability,safety,usability |
| 4 | Government and Military | Accuracy,confidentiality,performance,privacy,provability,reusability,security,standardizability,usability,verifiability,viability |
| 5 | Insurance | Accuracy,confidentiality,integrity,interoperability,security,usability |
| 6 | Medical/ Health Care | Communicativeness,confidentiality,integrity,performance,privacy,reliability,safety,security,traceability,usability |
| 7 | Telecommunication Services | Compatibility,conformance,dependability,installability,maintainability,performance,portability,reliability,usability |
| 8 | Transportation | Accuracy,availability,compatibility,completeness,confidentiality,dependability,integrity,performance,safety,security,verifiability |

**TABLE 5.** Type of systems & its relevant non-functional requirements, adopted from [47], [48].

| ID | System Type | Recommended NFRs |
|---|---|---|
| 1 | Real-time systems | Compatibility,Completeness,Conformance, extensibility, installability, portability, dependability, maintainability, integrity, accuracy, confidentiality, verifiability, availability, performance, security, usability, reliability, safety |
| 2 | Safety Critical systems | Communicativeness,Dependability,Safety, reliability, performance, security, usability, integrity |
| 3 | Web systems | Interoperability,Privacy,Scalability,performance, security, usability,integrity |
| 4 | Information systems | Provability,Reusability, Standardizability, reliability, accuracy, confidentiality, verifiability, availability, performance, security, usability, interoperability, privacy, traceability, viability |
| 5 | Process-controlled systems | maintainability, safety, reliability, availability, performance, security, usability |

To have better and clear recommendations, the tool uses the above information and provides three views of recommendation:

1) The first view is a recommendation from the type of domain.
2) The second view is a recommendation from the type of system.
3) The third view is historical data from previous projects in the tool.

### 2) REQUIREMENTS ELICITATION

One of the key activities in requirement engineering is the elicitation of requirements. It is the foundation for whether the project will continue successfully or not. Early in the project's lifecycle, non-functional requirements should be identified to help with project estimations of effort, size, and cost. This will help to reduce rework. [8]. Non-functional requirements can be identified in the project life cycle in one of the following stages [3]:

1) During sprint zero, resulting in the definition of a general software architecture.
2) During the initial stages of development.
3) During the initial iteration of the project, then refined to more details in later iterations.
4) During each iteration.
5) In the late stages of the project which causes a lot of re-work because non-functional requirements were not considered in the initial stages

In our proposed solution, the non-functional requirements are identified during the initial iteration of the project and then can be refined into more details later if needed. The following are the steps for non-functional requirements elicitation in our proposed solution:

1) The first step is where the product owner/business analyst accesses MANoR to receive the recommendations of non-functional requirements based on previous projects and pre-defined data which are based on research.
2) The second step is that the product owner/business analyst conducts a non-functional requirements session with the client. The following are the details of the session:
   (a) Description of the session: it is one session with a maximum duration of two hours to discuss the non-functional requirements of the software being developed.
   (b) Objectives of the session:
      (i) Explains the meaning of non-functional requirements in general.
      (ii) Describes the recommended non-functional requirements based on step 1 mentioned above.
      (iii) Shows the most common non-functional requirements mentioned in the literature.
      (iv) Shows the non-functional requirements conflicts mentioned in the literature.
   (c) Pre-requisites of the session:
      (i) Type of the domain for software being developed
      (ii) Type of application for software being developed
   (d) The input of the session:
      (i) List of recommended non-functional requirements for the software being developed
   (e) The output of the session:
      (i) List of non-functional requirements given by the client and to be further analyzed
   (f) People involved in the session:
      (i) Technical side: the product owner/business analyst will attend as he or she is responsible for the product backlog which includes non-functional requirements
      (ii) Client-side: key stakeholders of the system.

### B. MANoR: STAGE II

In this section, we discuss the next three components of the framework. The first component of Stage II is the requirements analysis which is discussed in section I. The second component is requirements documentation which is discussed in section II. The third component is requirements validation which is discussed in section III.

#### 1) REQUIREMENT ANALYSIS

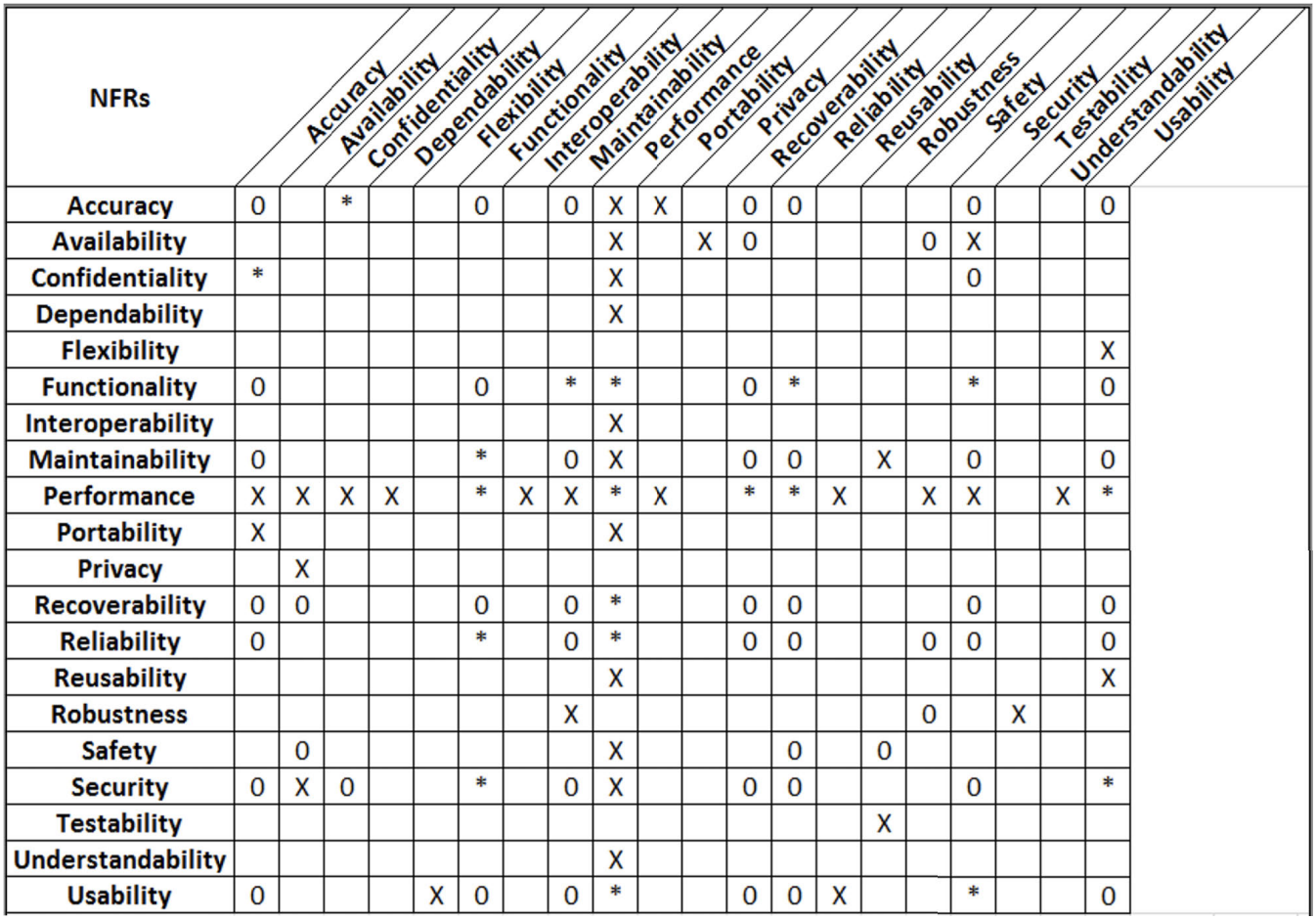
The main task is to determine whether the elicited requirement is clear, complete, unambiguous, and not contradicting. Conflicts mean interference or inconsistency between requirements (functional or non-functional requirements) [49]. Conflicts are resolved in this activity by negotiation with stakeholders. In our proposed solution, the focus is on the non-functional requirements. Conflicting non-functional requirements refer to situations in which meeting one requirement may have an impact on another [50]. For example, a specific module requires an extra security mechanism which increases the complexity of the module and therefore increases the difficulty of the user interaction [50]. This would cause a conflict between security and usability. Another example is between security and performance, where the user wants the specific functionality to be finished in a minimal time and in addition, specifies a high-security protocol [49]. Conflicts between requirements can occur for a variety of reasons, for example, in case there is a huge quantity of requirements, changes that may occur during software development, software with different stakeholders with different requirements or interests, or by adding a new stakeholder in the project [49]. Given how many individuals are engaged in software development, conflicting requirements are one of the main causes of software failure [51]. Between 40% and 60% of requirements conflict, according to a study of two-year multiple-project analysis [52], [53], [54]. Among those requirements are non-functional requirements that involve the greatest conflict which nearly takes half of the total requirements' conflicts [52], [53]. Figure 2 shows the conflict matrix between non-functional requirements where the relationship of conflict among non-functional requirements is presented in four ways [55], [50]:

1) Absolute conflict: labeled as "X" which means there is a conflict between two NFRs.
2) Relative conflict: labeled as "*" which means they are not always in conflict and it depends on the cases, therefore sometimes they are in conflict and at other times they are not.
3) Never in conflict: labeled as "0" which means there is no inherent conflict between the two NFRs.
4) Unknown conflict is labeled with blank space which means there is no information available in the literature about those sets of non-functional requirements.

Figure 2 is implemented in MANoR which provides details on which non-functional requirements have conflicts with each other. This helps the users of the tools during the analysis activity.

In general, two main factors can cause conflicts between non-functional requirements [53]:

1) The different needs and perspectives of different stakeholders.



| NFRs | Accuracy | Availability | Confidentiality | Dependability | Flexibility | Functionality | Interoperability | Maintainability | Performance | Portability | Privacy | Recoverability | Reliability | Reusability | Robustness | Safety | Security | Testability | Understandability | Usability |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Accuracy | 0 | | * | | | 0 | | 0 | X | X | | 0 | 0 | | | | 0 | | | 0 |
| Availability | | | | | | | | | X | | X | 0 | | | | 0 | X | | | |
| Confidentiality | * | | | | | | | | X | | | | | | | | 0 | | | |
| Dependability | | | | | | | | | X | | | | | | | | | | | |
| Flexibility | | | | | | | | | | | | | | | | | | | | X |
| Functionality | 0 | | | | | 0 | | * | * | | | 0 | * | | | | * | | | 0 |
| Interoperability | | | | | | | | | X | | | | | | | | | | | |
| Maintainability | 0 | | | | | * | | 0 | X | | | 0 | 0 | | X | | 0 | | | 0 |
| Performance | X | X | X | X | | * | X | X | * | X | | * | * | X | | X | X | | X | * |
| Portability | X | | | | | | | | X | | | | | | | | | | | |
| Privacy | | X | | | | | | | | | | | | | | | | | | |
| Recoverability | 0 | 0 | | | | 0 | | 0 | * | | | 0 | 0 | | | | 0 | | | 0 |
| Reliability | 0 | | | | | * | | 0 | * | | | 0 | 0 | | | 0 | 0 | | | 0 |
| Reusability | | | | | | | | | X | | | | | | | | | | | X |
| Robustness | | | | | | | | X | | | | | | | | 0 | | X | | |
| Safety | | 0 | | | | | | | X | | | | 0 | | 0 | | | | | |
| Security | 0 | X | 0 | | | * | | 0 | X | | | 0 | 0 | | | | 0 | | | * |
| Testability | | | | | | | | | | | | | | | X | | | | | |
| Understandability | | | | | | | | | X | | | | | | | | | | | |
| Usability | 0 | | | | X | 0 | | 0 | * | | | 0 | 0 | X | | | * | | | 0 |

**FIGURE 2.** Non-functional requirements Conflict Matrix, adopted from [50], [55].

2) The inherent contradiction between non-functional requirements is a result of specific characteristics of some non-functional requirements.

There are three main steps in managing conflicts between non-functional requirements [53]:

1) Conflict identification: the purpose is to identify the conflicts early during the software development life cycle
2) Conflict analysis: the purpose is to evaluate and investigate potential conflict
3) Conflict resolution: the purpose is to resolve the potential conflict

Given the above information, the tool helps the product owner to identify the conflicts which is step one. The tool provides the non-functional requirement conflict matrix to the end user. Then in step two, the product owner evaluates the conflict and then gets back to the client to resolve this conflict which is step three.

### 2) REQUIREMENTS DOCUMENTATION

One of the biggest challenges is the minimal documentation. Non-functional requirements documentation is part of that challenge as it should be documented along with the functional requirements. The question is how to document non-functional requirements in Agile software development. Is it better to document it as a separate user story or as a user story parameter? In our proposed solution, a separate user story will be used for each non-functional requirement. Separate user stories for non-functional requirements would be better in the product backlog, instead of having a user story with a non-functional parameter [56]. This will help because most non-functional requirements can be split into different sprints [56]. It is better to create separate user stories for the non-functional requirements in the product backlog, instead of having non-functional requirement parameters in the user story [56]. There are different views for the user stories. The following list explains the different views of the user stories in our proposed solution which is implemented in the tool developed specifically for this solution:

1) Non-functional user story vs. functional user story: The user story can contain functional or non-functional requirements in our proposed solution. The tool allows the user to select the type of requirement and based on the selection, the tool will provide proper fields. An example of a non-functional requirement user story in MANoR is ''performance'', its description is ''the response time shall be one second'', related to the field ''Registration Module''.

**TABLE 6.** Summary of the results.

| No | Items | Results |
|---|---|---|
| 1 | Number of total respondents that filled the evaluation form | 52 total respondents |
| 2 | Number of technical respondents that have current jobs in software companies | 44 technical respondents |
| 3 | Number of participating software companies | 19 software companies |
| 4 | Number of academic respondents that worked in a university | 8 academic respondents |
| 5 | Number of participating universities | 3 universities |
| 6 | Number of participating countries | 3 countries |
| 7 | Number of respondents based on type of team | 11 respondents in the management team<br>11 respondents in the testing team<br>10 respondents in the development team<br>10 respondents in the analysis team<br>8 respondents in the academic team<br>1 respondent in the database and architecture team<br>1 respondent in the product management team |

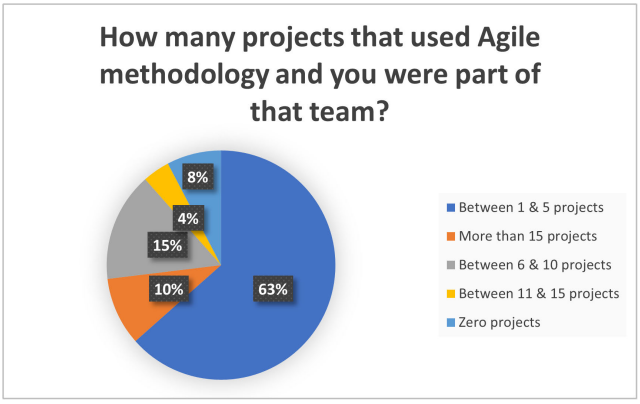



**FIGURE 3.** Technical experience in agile methodology.

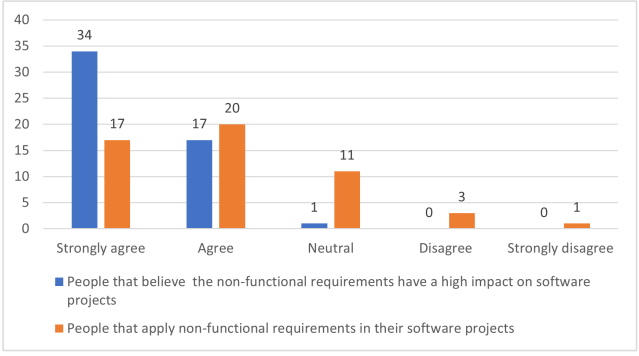



**FIGURE 4.** Importance of non-functional requirements.

2) Standard user story vs. custom user story: A standard user story is the normal user story that is used in Agile. A normal user story is written as the following template: *As a <type of user> I want <specific task> so that <achieve some value/goals>*. On the other hand, the custom user story is the traditional way of writing any business requirement. Basically, it is one sentence without applying any templates or rules. For example, students can register courses online. The tool allows the user to select how the user wants to write the user story: standard or custom.
3) System vs. module vs. epic vs. functional requirement: In our proposed solution, any non-functional requirement is related either to a system, module, epic, or functional requirement. This means from early development the analyst will know which non-functional requirements affect the entire system. Or that it affects a specific module or an epic or if it is related to a very specific functional requirement written in the form of a user story.

### 3) REQUIREMENTS VALIDATION

This part of the proposed solution is where validation of the non-functional requirements is done. In other words, making sure that both the client and the technical team are on the same page regarding the non-functional requirements for the given software. The technical team confirms the non-functional requirements with the client. Once the validation is done with the client, the normal process of Scrum will take over.

## V. VALIDATION & RESULTS

The framework was validated using an evaluation form. The form consists of two main parts, the first part targets the person's background which includes technical experience in the software development domain, Agile, and Scrum. The second part targets the person's view on the framework. Both parts consist of twenty-one different questions which include short answer questions and the standard Likert-scale questions. Fifty-two different people filled in the evaluation form. The process of selecting those people depended on several factors which include the following:

1) There are two main categories of people that filled in the form: academics and technical staff. The academic team involves instructors working in different universities. The technical team involves different technical people working in a number of different companies. Including both the academic team and technical team in the evaluation, is meant to solicit substantive input from both worlds: research and practice.
2) The technical team includes precisely forty-four different people working in nineteen different companies in three different countries as shown in table 6. The technical team involves different roles/titles working in different departments such as analysis, development, testing, and management. Different people per

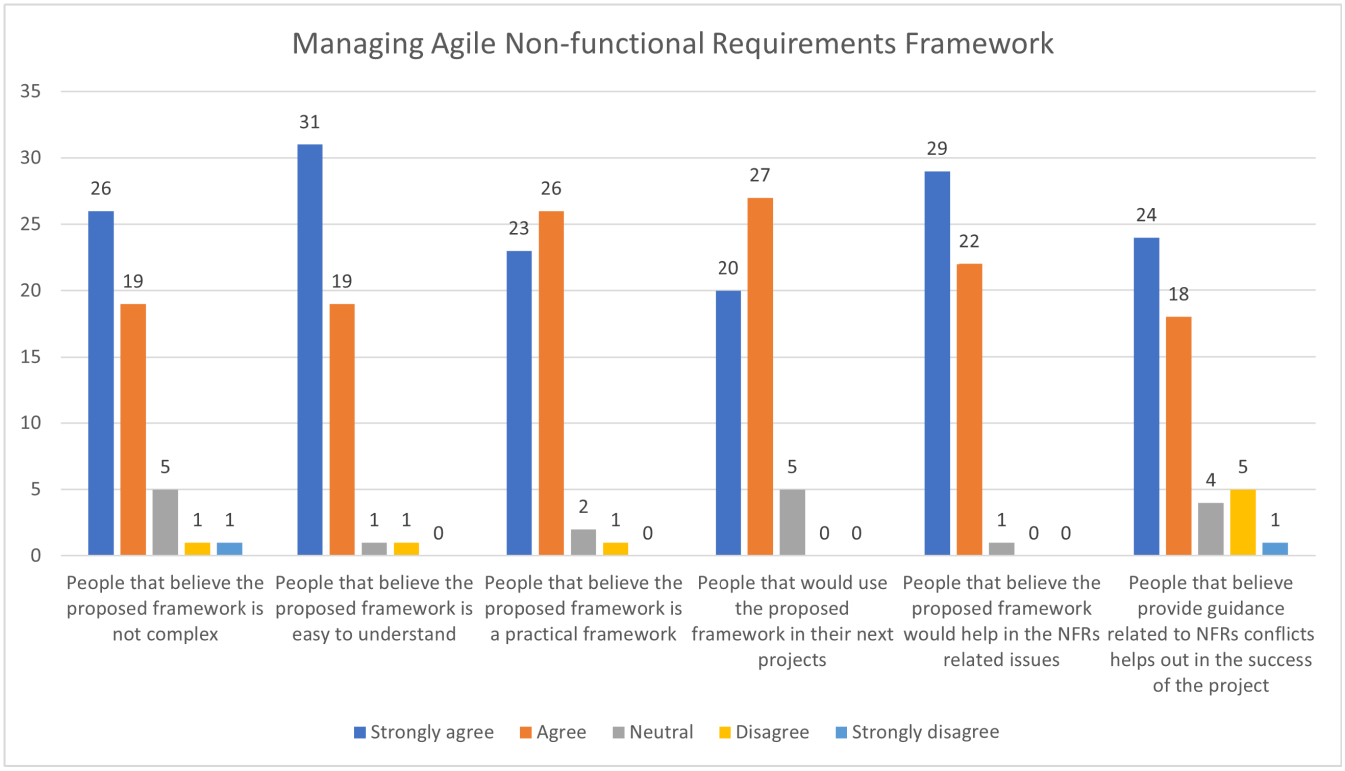



**FIGURE 5.** The impact of the MANoR framework.

department were involved in evaluating the framework for each department. Involving different departments is meant to provide different perspectives from different roles in software development.

3) The academic team includes eight different instructors working in three different universities as shown in table 6. Those instructors are working in the Software Engineering Department of those universities. The reason for selecting this department is that Agile development is one of the major research areas in those universities.

Figure 3 shows the technical experience in the Agile development process of the people who filled in the evaluation form by showing the number of Agile projects they worked in. As shown in Figure 3, there are 63% of the people did work in between one and five Agile projects, in another word, this means 33 out of 52 people. In addition, 17% of people did work between six and ten Agile projects, in another word, this means 8 out of 52 people.

Figure 4 shows two important questions regarding the non-functional requirements. The first question references the high impact of non-functional requirements in software projects. The answers are positive as 34 people strongly agree and 17 people do agree that non-functional requirements have a high impact on the success of the projects. In other words, 51 out of 52 people think non-functional requirements are important. The second question references the reality of applying non-functional requirements in their software projects. As there are 17 different people who strongly agree and 20 different people who agree. This means a total of 37 out of 49 people do consider non-functional requirements in their projects. Comparing the answers to the last two questions, 51 people believe non-functional requirements are important, however, only 37 of those 52 do consider non-functional requirements in their projects.

Figure 5 focuses on the framework "MANoR" by showing six different questions answered by both teams. Each question is a multiple-choice Likert-scale question. The purpose of those questions is to evaluate the framework and see whether it helps in the issues regarding the non-functional requirements or not. The first question targets complexity and 45 people agree that it is simple which is an important point because one of the principles of Agile is simplicity. The second question targets the understandability of the framework and it shows that 50 people find the framework easy to understand. The third and fourth questions target the realism of the framework, and whether people would use it in future projects, respectively. The answers show that 49 people agree that the framework is practical and realistic and 47 people agree that it can be used in their future projects. The fifth question targets whether the framework helps in the issues related to non-functional requirements such as the elicitation, recommendation, analysis, documentation, and validation of non-functional requirements. In this question, a total of 51 agree that the framework helps in those issues which also shows a positive impact of the framework. The sixth question targets the idea of non-functional requirement conflicts. 42 people agree that uncovering the conflicts early

on in the project helps in reducing the risk of having residual non-functional requirements conflicts.

The proposed framework was based on the principles and values of the Agile Manifesto and therefore offers a flexible way to handle non-functional requirements in Agile. The above statistics and results show the positive impact of the framework on dealing with non-functional requirements in Agile.

## VI. CONCLUSION & FUTURE WORKS

Ignoring non-functional requirements in Agile Software Development greatly impacts the software's success. Therefore, there is a need to focus on the problems of ignoring non-functional requirements in Agile Software Development and accordingly provide a lightweight solution while keeping the principles and values of Agile Software Development intact. The proposed framework "MANoR" focuses on non-functional requirements in the activities of elicitation, recommendation, analysis, documentation, and validation in Agile Software Development. Finally, the proposed framework was validated by technical experts and an academic team to prove its objective of focusing on non-functional requirements in Agile without breaking the principles and values of Agile. Future work will include the validation of the framework through real-life case studies. By implementing such a framework, companies will focus on the importance and impact of non-functional requirements in Agile projects. Therefore, it is expected to improve the quality of the resultant software and reduce the risk of profit loss, effort rework, or even project failure.

## REFERENCES

[1] A. Jarzebowicz and P. Weichbroth, "A systematic literature review on implementing non-functional requirements in agile software development: Issues and facilitating practices," in *Lean and Agile Software Development*, vol. 408. Cham, Switzerland: Springer, 2021.

[2] W. Helmy and G. H. Galal-edeen, *Managing Non-Functional Requirements*. Cham, Switzerland: Springer, 2022.

[3] A. Jarzebowicz and P. Weichbroth, "A qualitative study on non-functional requirements in agile software development," *IEEE Access*, vol. 9, pp. 40458–40475, 2021, doi: 10.1109/ACCESS.2021.3064424.

[4] W. Behutiye, P. Karhapää, L. López, X. Burgués, S. Martínez-Fernández, A. M. Vollmer, P. Rodríguez, X. Franch, and M. Oivo, "Management of quality requirements in agile and rapid software development: A systematic mapping study," *Inf. Softw. Technol.*, vol. 123, Jul. 2020, Art. no. 106225, doi: 10.1016/j.infsof.2019.106225.

[5] W. Behutiye, P. Karhapää, D. Costal, M. Oivo, and X. Franch, "Non-functional requirements documentation in agile software development: Challenges and solution proposal," 2017, *arXiv:1711.08894*.

[6] W. Alsaqaf, M. Daneva, and R. Wieringa, "Agile quality requirements engineering challenges: First results from a case study," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Nov. 2017, pp. 454–459, doi: 10.1109/ESEM.2017.61.

[7] D. S. Cruzes, M. Felderer, T. D. Oyetoyan, M. Gander, and I. Pekaric, "How is security testing done in agile teams? A cross-case analysis of four software teams," in *Proc. 8th Int. Conf. Agile Processes Softw. Eng. Extreme Program.*, 2019, pp. 133–134, doi: 10.18420/se2019-40.

[8] A. Silva, P. Pinheiro, A. Albuquerque, and J. Barroso, "A process for creating the elicitation guide of non-functional requirements," *Adv. Intell. Syst. Comput.*, vol. 465, pp. 293–302, Jan. 2016, doi: 10.1007/978-3-319-33622-0_27.

[9] M. Younas, D. N. A. Jawawi, I. Ghani, and M. A. Shah, "Extraction of non-functional requirement using semantic similarity distance," *Neural Comput. Appl.*, vol. 32, no. 11, pp. 7383–7397, Jun. 2020, doi: 10.1007/s00521-019-04226-5.

[10] A. Aldave, J. M. Vara, D. Granada, and E. Marcos, "Leveraging creativity in requirements elicitation within agile software development: A systematic literature review," *J. Syst. Softw.*, vol. 157, Nov. 2019, Art. no. 110396, doi: 10.1016/j.jss.2019.110396.

[11] J. M. Fernandes and M. Almeida, "Classification and comparison of agile methods," in *Proc. 7th Int. Conf. Quality Inf. Commun. Technol.*, Sep. 2010, pp. 391–396, doi: 10.1109/QUATIC.2010.71.

[12] A. Silva, T. Araújo, J. Nunes, M. Perkusich, E. Dilorenzo, H. Almeida, and A. Perkusich, "A systematic review on the use of definition of done on agile software development projects," in *Proc. 21st Int. Conf. Eval. Assessment Softw. Eng.*, Jun. 2017, pp. 364–373, doi: 10.1145/3084226.3084262.

[13] E. M. Schön, D. Winter, M. J. Escalona, and J. Thomaschewski, "Key challenges in agile requirements engineering," in *Proc. 18th Int. Conf. Agile Processes Softw. Eng. Extreme Program.*, vol. 283, 2017, pp. 37–51, doi: 10.1007/978-3-319-57633-6_3.

[14] V. Gaikwad and P. Joeg, "A case study in requirements engineering in context of agile," *Int. J. Appl. Eng. Res.*, vol. 12, no. 8, pp. 1697–1702, 2017.

[15] K. Elghariani and N. Kama, "Review on agile requirements engineering challenges," in *Proc. 3rd Int. Conf. Comput. Inf. Sci. (ICCOINS)*, Aug. 2016, pp. 507–512, doi: 10.1109/ICCOINS.2016.7783267.

[16] S. Alam, S. Nazir, S. Asim, and D. Amr, "Impact and challenges of requirement engineering in agile methodologies: A systematic review," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 4, pp. 411–420, 2017, doi: 10.14569/ijacsa.2017.080455.

[17] M. Käpyaho and M. Kauppinen, "Agile requirements engineering with prototyping: A case study," in *Proc. IEEE 23rd Int. Requirements Eng. Conf.*, Aug. 2015, pp. 334–343, doi: 10.1109/RE.2015.7320450.

[18] I. Inayat, L. Moraes, M. Daneva, and S. S. Salim, "A reflection on agile requirements engineering: Solutions brought and challenges posed," in *Proc. Sci. Workshop*, May 2015, pp. 1–12, doi: 10.1145/2764979.2764985.

[19] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Comput. Hum. Behav.*, vol. 51, pp. 915–929, Oct. 2015, doi: 10.1016/j.chb.2014.10.046.

[20] D. Sunner and H. Bajaj, "Classification of functional and non-functional requirements in agile by cluster neuro-genetic approach," *Int. J. Softw. Eng. Appl.*, vol. 10, no. 10, pp. 129–138, Oct. 2016, doi: 10.14257/ijseia.2016.10.10.13.

[21] M. Saleh, F. Baharom, S. Farvin, P. Mohamed, and M. Ahmad, "A systematic literature review of challenges and critical success factors in agile requirement engineering," in *Proc. 9th Knowl. Manag. Int.*, Jul. 2018, pp. 248–254.

[22] M. Batra and A. Bhatnagar, "A research study on critical challenges in agile requirements engineering," *Int. Res. J. Eng. Technol.*, vol. 10, pp. 1214–1219, Jun. 2019.

[23] R. Telesko, "Road to agile requirements engineering: Lessons learned from a web app project," *Stud. Syst. Decis. Control*, vol. 141, pp. 65–78, 2018, doi: 10.1007/978-3-319-74322-6_5.

[24] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: An empirical study," *Inf. Syst. J.*, vol. 20, no. 5, pp. 449–480, Nov. 2007, doi: 10.1111/j.1365-2575.2007.00259.x.

[25] M. S. Et. al, "Critical success factors and challenges in agile requirements engineering," *Turkish J. Comput. Math. Educ.*, vol. 12, no. 3, pp. 1670–1682, Apr. 2021, doi: 10.17762/turcomat.v12i3.989.

[26] W. M. Farid, "The NORMAP methodology: Lightweight engineering of non-functional requirements for agile processes," in *Proc. 19th Asia–Pacific Softw. Eng. Conf.*, Dec. 2012, pp. 322–325, doi: 10.1109/APSEC.2012.23.

[27] D. Domah and F. J. Mitropoulos, "The NERV methodology: A lightweight process for addressing non-functional requirements in agile software development," in *Proc. SoutheastCon*, Apr. 2015, pp. 1–7, doi: 10.1109/SECON.2015.7133028.

[28] S. Dragicevic, S. Celar, and L. Novak, "Use of method for elicitation, documentation, and validation of software user requirements (MEDoV) in agile software development projects," in *Proc. 6th Int. Conf. Comput. Intell., Commun. Syst. Netw.*, May 2014, pp. 65–70, doi: 10.1109/CICSyN.2014.27.

[29] R. R. Maiti and F. J. Mitropoulos, "Capturing, eliciting, and prioritizing (CEP) NFRs in agile software engineering," in *Proc. SoutheastCon*, Mar. 2017, pp. 1–7, doi: 10.1109/SECON.2017.7925365.

[30] R. R. Maiti, A. Krasnov, and D. M. Wilborne, "Agile software engineering & the future of non-functional requirements," *J. Softw. Eng. Pract.*, vol. 2, no. 1, pp. 1–8, 2018.

[31] S. Jeon, M. Han, E. Lee, and K. Lee, "Quality attribute driven agile development," in *Proc. 9th Int. Conf. Softw. Eng. Res., Manage. Appl.*, Aug. 2011, pp. 203–210, doi: 10.1109/SERA.2011.24.

[32] S. Kopczynska and J. Nawrocki, "Using non-functional requirements templates for elicitation: A case study," in *Proc. IEEE 4th Int. Workshop Requirements Patterns*, Aug. 2014, pp. 47–54, doi: 10.1109/RePa.2014.6894844.

[33] W. M. Farid and F. J. Mitropoulos, "NORMATIC: A visual tool for modeling non-functional requirements in agile processes," in *Proc. IEEE Southeastcon*, Mar. 2012, pp. 1–8, doi: 10.1109/SECon.2012.6196989.

[34] W. M. Farid and F. J. Mitropoulos, "NORPLAN: Non-functional requirements planning for agile processes," in *Proc. IEEE Southeastcon*, Apr. 2013, pp. 1–12, doi: 10.1109/SECON.2013.6567463.

[35] W. M. Farid and F. J. Mitropoulos, "Visualization and scheduling of non-functional requirements for agile processes," in *Proc. IEEE Southeastcon*, Apr. 2013, pp. 1–15, doi: 10.1109/SECON.2013.6567413.

[36] R. R. Maiti and F. J. Mitropoulos, "Capturing, eliciting, predicting and prioritizing (CEPP) non-functional requirements metadata during the early stages of agile software development," in *Proc. SoutheastCon*, Apr. 2015, pp. 1–8, doi: 10.1109/SECON.2015.7133007.

[37] F. Ramos, A. Pedro, M. Cesar, A. Costa, M. Perkusich, H. Almeida, and A. Perkusich, "Evaluating software developers' acceptance of a tool for supporting agile non-functional requirement elicitation," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, Jul. 2019, pp. 26–31, doi: 10.18293/SEKE2019-107.

[38] F. Ramos, A. A. M. Costa, M. Perkusich, H. Almeida, and A. Perkusich, "A non-functional requirements recommendation system for scrum-based projects," in *Proc. Int. Conf. Softw. Eng. Knowl. Eng.*, Jul. 2018, pp. 149–154, doi: 10.18293/SEKE2018-107.

[39] V. Gaikwad, P. Joeg, and S. Joshi, "AgileRE: Agile requirements management tool," *Adv. Intell. Syst. Comput.*, vol. 661, pp. 236–249, Jan. 2018, doi: 10.1007/978-3-319-67618-0_22.

[40] M. Younas, D. N. Jawawi, I. Ghani, and R. Kazmi, "Non-functional requirements elicitation guideline for agile methods," *J. Telecommun., Electron. Comput. Eng.*, vol. 9, nos. 3–4, pp. 137–142, Oct. 2017.

[41] H. Saeeda, J. Dong, Y. Wang, and M. A. Abid, "A proposed framework for improved software requirements elicitation process in SCRUM: Implementation by a real-life Norway-based IT project," *J. Softw., Evol. Process*, vol. 32, no. 7, pp. 1–24, Jul. 2020, doi: 10.1002/smr.2247.

[42] W. M. Farid and F. J. Mitropoulos, "Novel lightweight engineering artifacts for modeling non-functional requirements in agile processes," in *Proc. IEEE Southeastcon*, Mar. 2012, pp. 1–7, doi: 10.1109/SECon.2012.6196988.

[43] C. Werner, Z. S. Li, N. Ernst, and D. Damian, "The lack of shared understanding of non-functional requirements in continuous software engineering: Accidental or essential?" in *Proc. IEEE 28th Int. Requirements Eng. Conf. (RE)*, Aug. 2020, pp. 90–101, doi: 10.1109/RE48521.2020.00021.

[44] V. Bajpai and R. P. Gorthi, "On non-functional requirements? A survey," in *Proc. IEEE Students' Conf. Elect., Electron. Comput. Sci.*, 2012, pp. 1–3.

[45] R. R. Maiti and F. J. Mitropoulos, "Prioritizing non-functional requirements in agile software engineering," in *Proc. SouthEast Conf.*, Apr. 2017, pp. 212–214, doi: 10.1145/3077286.3077565.

[46] M. Younas, D. N. A. Jawawi, M. A. Shah, A. Mustafa, M. Awais, M. K. Ishfaq, and K. Wakil, "Elicitation of nonfunctional requirements in agile development using cloud computing environment," *IEEE Access*, vol. 8, pp. 209153–209162, 2020, doi: 10.1109/ACCESS.2020.3014381.

[47] D. Mairiza, D. Zowghi, and N. Nurmuliani, "An investigation into the notion of non-functional requirements," in *Proc. ACM Symp. Appl. Comput.*, Mar. 2010, pp. 311–317, doi: 10.1145/1774088.1774153.

[48] D. Mairiza and D. Zowghi, "Constructing a catalogue of conflicts among non-functional requirements," in *Evaluation of Novel Approaches to Software Engineering* (Communications in Computer and Information Science), vol. 230, 2011, pp. 31–44, doi: 10.1007/978-3-642-23391-3_3.

[49] M. Aldekhail, A. Chikh, and D. Ziani, "Software requirements conflict identification: Review and recommendations," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 10, pp. 1–16, 2016, doi: 10.14569/ijacsa.2016.071044.

[50] R. M. Carvalho, "Dealing with conflicts between non-functional requirements of UbiComp and IoT applications," in *Proc. IEEE 25th Int. Requirements Eng. Conf. (RE)*, Sep. 2017, pp. 544–549, doi: 10.1109/RE.2017.51.

[51] M. N. A. Chaudhary, N. Sabahat, and S. K. Toor, "RES-TOOL to overcome requirements elicitation barriers in Pakistan software industry," in *Proc. Int. Conf. Electr., Commun., Comput. Eng. (ICECCE)*, Jun. 2020, pp. 1–6, doi: 10.1109/ICECCE49384.2020.9179321.

[52] D. Mairiza, D. Zowghi, and V. Gervasi, "Utilizing TOPSIS: A multi criteria decision analysis technique for non-functional requirements conflicts," in *Requirements Engineering* (Communications in Computer and Information Science), vol. 432, 2014, pp. 31–44, doi: 10.1007/978-3-662-43610-3_3.

[53] D. Mairiza, D. Zowghi, and N. Nurmuliani, "Managing conflicts among non-functional requirements," in *Proc. Austral. Workshop Requirements Eng.*, 2009, pp. 11–19.

[54] D. Mairiza, D. Zowghi, and N. Nurmuliani, "Towards a catalogue of conflicts among non-functional requirements," in *Proc. 5th Int. Conf. Eval. Novel Approaches Softw. Eng.*, 2010, pp. 20–29, doi: 10.5220/0002927900200029.

[55] F. Pinciroli, "Improving software applications quality by considering the contribution relationship among quality attributes," *Proc. Comput. Sci.*, vol. 83, pp. 970–975, Jan. 2016, doi: 10.1016/j.procs.2016.04.194.

[56] A. E. Sabry and S. S. El-Rabbat, "Proposed framework for handling architectural NFR's within scrum methodology," in *Proc. Int. Conf. SERP*, 2015, p. 238.

[57] M. A. Subih, B. Hayat, I. Mazhar, A. Yousaf, M. Usman, T. Wakeel, and W. Ali, "Comparison of agile method and scrum method with software quality affecting factors," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 5, pp. 531–535, 2019, doi: 10.14569/ijacsa.2019.0100569.

**EZELDIN SHERIF** received the B.Sc. degree in software engineering from the Faculty of Computer Science and Information Technology, Ahram Canadian University, Cairo, Egypt, in 2009, and the Master of Science (M.Sc.) degree in communication and information technology from the Software Engineering Department, Nile University, Cairo, in 2014. He was a Teaching Assistant with Ahram Canadian University, from 2009 to 2014, where he has been an Assistant Lecturer, since 2014. His research interests include software engineering, requirements engineering, project management, and agile software development.

**WALEED HELMY** received the B.Sc., Master of Science (M.Sc.), and Ph.D. degrees in information systems from the Faculty of Computers and AI, Cairo University, Cairo, Egypt, in 2003, 2007, and 2013, respectively. He was a Teaching Assistant with Cairo University, from 2003 to 2007, where he was also an Assistant Lecturer, from 2007 to 2013. He has been an Associate Lecturer with Cairo University, since 2013. His research interests include data engineering, software engineering, software architecture design, and requirements engineering.

**GALAL HASSAN GALAL-EDEEN** received the B.Sc. degree in management sciences (computing) from Sadat Academy for Management Sciences, Cairo, Egypt, the M.Sc. degree in systems analysis and design from City University, London, the Ph.D. degree in information systems engineering from Brunel University, London, the B.A. degree in architecture from the University of Greenwich, U.K., and the M.Sc. degree in advanced architectural studies from University College London, U.K. From 1988 to 2002, he was an assistant professor, an associate professor-equivalent ranks, and a senior research fellow in several U.K. universities. He served as a senior consultant in innovation and the chief innovation policy advisor for government. He is currently a Professor in information systems with The American University in Cairo. His research interests include systems architecting, software engineering methodologies, usability evaluation, healthcare systems, and human–computer interaction. He won research funding from the European Union, USAid, EPSRC, DAAD, ITIDA, UNDP, and the NAIS-Lorentz Centre (the Netherlands). He is the author or coauthor of over 125 refereed articles and serves on the EAB of three international journals. He is a Full Professional Member of the British Computer Society and the American Computing Machinery.