Analyzing Non-Functional Requirements (NFRs) beyond Requirements Engineering

Cyrille Dongmo
Computer Science Department, School of Computing, CSET, Science Campus, University of South Africa Unisa, Florida Park, South Africa
dongmc@unisa.ac.za (corresponding author)

Received: 2 December 2024 | Revised: 27 December 2024 and 8 January 2025

 DOI: https://doi.org/10.48084/etasr.9800

ABSTRACT

The development of Non-Functional Requirements (NFRs) alongside functional ones has long been the concern of both researchers and software engineers. The main purpose is to derive means to propagate the influence of NFRs throughout the different phases of the software development process. Despite the important progress made through Goal-Oriented Requirements Engineering (GORE) techniques, in terms of requirements elicitation, analysis, and operationalization of goals specifying NFRs, the inherent non-deterministic character of NFRs makes it very hard to anticipate at the requirements phase their impact on subsequent development phases. Thus, at a given phase of development, the actions required to satisfy an NFR, known as Non-Functional actions (NF-actions), depend on various factors, including the actions performed at the previous step as well as the type and nature of the object influenced by the NFR. This study proposes the concept of Complementary Non-Functional Actions (CNF-Actions) to facilitate the analysis of NFRs throughout the Software Development Life Cycle (SDLC). The application of the concept to a theoretical case study of an electronic voting system demonstrated its ability to facilitate the analysis of NFRs while developing the functional requirements and, therefore, contributing to extending the analysis of NFRs to other software phases of the SDLC.

Keywords-non-functional requirements; software development life cycle; non-functional requirements analysis; goal-oriented requirement language; use case maps

I. INTRODUCTION

Software development is a continuous and complex process that aims to transform the initial user needs or requirements into a tested correct software product. Each paradigm or method subdivides the process into a certain number of phases for which specific activities and supporting tools are suggested. The traditional Software Development Lifecycle (SDLC) proposes four major development phases [1]. Although different authors use different terminologies to designate each phase, this study considers the requirements phase, the design phase, the development phase, and the implementation and maintenance phase. Independently of any software paradigm, the input to a software development phase is the output from the previous phase. Except for the requirements phase, in which input is from various sources, including, among others, stakeholders, business processes, policy documents, etc. Therefore, the activities of each phase are conceptualized and defined to transform intermediate models and/or specifications from the previous phase into other intermediate models that constitute the deliverables for the phase and are in most cases more refined and detailed.

To date, numerous methods with well-defined processes and tools have been proposed, mainly for the development of functional requirements [1], which describe the services that the system in development should provide. Some examples include the Unified Modeling Language (UML) [2], the agile method [3], and formal methods such as the Z notation [4], none of which naturally integrate mechanisms for the analysis of Non-Functional Requirements (NFRs). NFRs define the quality, user interfaces, and constraints on how services are to be produced and rendered [1, 5]. With the growing complexity of software products, and hence the demand for quality, similar progress is required for the development of NFRs. To this end, remarkable efforts have been made in the engineering of NFRs at the requirements phase of the SDLC. However, beyond this phase, the design and development phases seem to simply inherit decisions made during the Requirements Engineering (RE) stage without further analysis. Mechanisms for the analysis of NFRs during the design and implementation of software are very scarce [6]. In general, concerns about NFRs reappear during product testing and operations, for instance, to establish the extent to which the Quality of Service (QoS) or NFR parameters, as agreed in contract documents, have been met.

A. NFR Development in Software Engineering

About two decades ago, the pressing need for scientific computation of NFRs was emphasized [7]. To date, research on NFR analysis processes beyond the requirements phase still requires more effort. Work on NFRs beyond the requirements phase is rarely process-oriented, pertaining to a step-by-step

analysis of NFRs during the design, development, and implementation phases of a software product. They focus either on supporting architectural decisions and documentation [8] or defining metrics and means to validate the final software product for the expected quality.

In [9], a lexical-based approach was proposed to integrate NFRs into ER and OO conceptual models. An NFR is selected from an NFR graph and carefully integrated into an ER or OO diagram specifying FRs, initially extended to allow the representation of the input NFRs. This approach requires the software designer to ensure that the model/diagram in which an NFR is integrated satisfies the NFR and has the expected qualities. This approach has the merit of bringing the analysis of NFRs from the requirements phase to the design of conceptual models. However, it is hard to tell what happens when the models with NFRs are refined or when development moves to the next phase.

In [10], a language was proposed to describe NFRs, called ProcessNFL, which follows a process-oriented approach to treat them. ProcessNFL defines three fundamental abstract elements for NFRs that provide it with the ability to explicitly model the relationships between NFRs and the implementation elements that affect them. These abstract elements are the Non-Functional Attribute (NF-Attribute), the Non-Functional Action (NF-Action), and the Non-Functional Property (NF-Property). An NF-Attribute describes the (measurable or not) characteristic of the NFR expected from the software product. On the other hand, an NF-Action models any action that affects the realization of the NF-Attribute and NF-Property, and specifies constraints over the NF-Attribute. NF-Action also defines the means for associate the analysis of NFRs with the development of functional requirements. In [11], an architecture for decision-making with an associated (generic) decision algorithm based on NFRs was proposed to identify the best available application configuration (BSolution) to satisfy NFRs in a dynamic software product line environment.

GORE methods, including, among others, the NFR framework [12], iStar [13], KAOS [14], GRL [15-16], TROPOS [17], etc., have constituted a compelling basis for the integration of GR and NFR analysis at the requirements level. They propose an environment for the elicitation, specification, decomposition, operationalization, and evaluation of the best solution for the realization of FRs and NFRs in the same model, namely the Softgoals Interdependency Graph (SIG). However, in terms of processes, they are limited when software development spans other SDLC phases. It is not possible to imagine all possible actions to realize NFRs in the requirements engineering stage, as some opportunities may only show up in some stage of development when some artifacts are produced.

The UML profile has also been widely exploited to develop methods for NFR analysis, bringing together the modeling and reasoning about NFRs and FRs. Two examples of such methods are OMG-standardized modeling languages, namely, the Systems Modeling Language (SysML) [18-19] and MARTE [19]. More detailed research to establish the current state-of-the-art in the development of NFRs throughout SDLC can be found in [6].

B. Research Problem and Question

A recent systematic literature review on the analysis of NFRs throughout the SDLC shows that to date, the efforts of most researchers and practitioners to develop NFRs fall primarily within the RE phase, where solutions or strategies produced to satisfy NFRs, generally functional in nature, merge with FRs [6]. Consequently, this shifts the focus to the development of FRs during the SDLC phases beyond RE, with little or no further analysis of NFRs until the final SDLC phase, where the system is tested and validated. The need to perform most of the FR design and transformation activities without explicit or formal consideration of NFRs further contributes to producing low-quality software, as it is practically impossible to imagine all possible NFR solutions in the requirements phase. More importantly, quality must be ensured in each phase of software development [20]. Thus, the following research question emerges:

Can a process or a mechanism be derived to continuously analyze and reason about NFRs throughout the SDLC phases beyond the requirement engineering phase?

The main contribution of this research includes the following:

- The concept of CNF-Action to stimulate and guide the analysis of NFRs throughout the SDLC beyond the requirements engineering phase,
- The application of the CNF-Action concept to a practical case study,
- The elicitation of future works that need to be carried out to further develop the CNF-Action concept.

II. RESEARCH METHOD

The research method is discussed in line with the research onion [21] and adapted in Figure 1. The research onion is followed from the outmost layer towards the innermost one. Thus, starting with the outer layer, the research philosophy is pragmatism in the first place but also integrates some aspects of interpretivism. The pragmatism results from the nature of the study contribution, which is to produce a practical means to further analyze NFRs in the SDLC beyond the requirements phase. The pseudo-interpretivism is revealed by the nature of the proposed solution, where at each step of software development, the perception and interpretation of NFRs, by the designer, may change depending on the software models considered. For example, the interpretation of performance or security when designing user interfaces is not necessarily the same as when designing class diagrams or deployment diagrams.

The approach to theory development at the second layer is essentially inductive and adductive to some extent. The reason is that the contribution of this article (which intends to be a practical solution) is applied to a case study to make observations that could lead, in future endeavors, to the development of a more comprehensible framework, hence, a generalized solution. The methodological choice, the third layer, is qualitative monomethod due to the use of a case study to evaluate the applicability and effectiveness of the suggested

complementary CNF-Action concept. Therefore, the strategy in the fourth layer combines action research and a case study. The time horizon is cross-sectional since this research was conducted at a specific point in time and not over several years, during which study subjects evolve. Procedures and techniques at this stage tend to data collection and analysis.

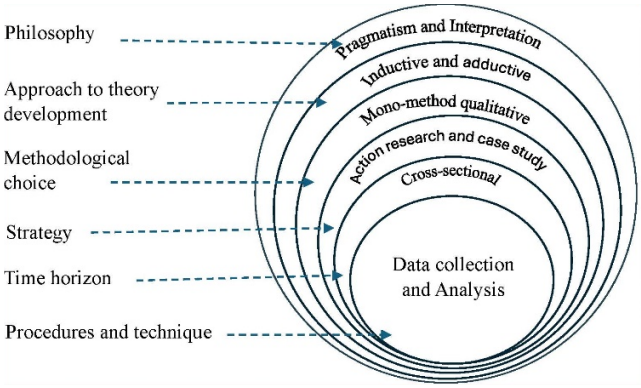

Fig. 1. Adapted from the research onion [20].

III. OVERVIEW OF GRL AND UCM

This section introduces the two methods used in the case study, namely, GRL and UCMs.

A. Goal-oriented Requirement Language (GRL)

GRL is one of the two components of the User Requirements Notation (URN) [15], dedicated to the elicitation and specification of FRs and NFRs. GRL is one well-established GORE that uses various concepts to model role players and different types of requirements, as well as the relationships between them. It is a visual modeling notation that uses visual symbols to construct a SIG that can be analyzed and evaluated using various evaluation algorithms [3].

1) GRL Basic Elements

The basic elements of GRL include goals to describe FRs, *Softgoals* to describe NFRs, *Tasks* to model generic activities or services, *Resources* to specify the resources needed to achieve a goal or to perform a task, *Actor* to specify a role player, and *Belief* to describe common knowledge that may be used as a guide to make a decision or a choice. The relationship between the basic elements of GRL, as well as the influence of such elements on one another, are specified using GRL link elements.

2) GRL Link Elements

The three most utilized types of GRL link elements are: Decomposition link used to model the And/Or decomposition of basic elements (such as goals, softgoals, tasks, and resources into more refined elements) to bring more details into the GRL model. As its name indicates, the Dependency link models the dependency relationship between GRL basic elements, while the Contribution link specifies the influence of an element on another.

3) GRL Contribution Types

The different contribution types a GRL element can have on others are: *Make* to indicate a positive and sufficient contribution, *Help* to indicate a positive contribution that is not sufficient, *Some Positive* to indicate when the contribution is positive but its extent is unknown, *Unknown* to indicate a contribution for which the extent and degree (positive or negative) is not known, *Some Negative* to indicate a negative contribution with unknown extent of contribution, *Break* to indicate when the contribution is negative and sufficient, and *Hurt* to indicate when the contribution is negative but not sufficient.

B. Use Case Maps (UCM)

UCM is a standardized scenario-based requirement notation technique that uses simple graphical elements or symbols to construct the system architecture in a map-like diagram [15]. UCM employs various concepts and building blocks to model service functionalities with the required resources, structure complex, and distributed systems, and specify any other artifacts, including components, agents, timers, and plugins [22-23].

1) Basic UCM Elements

Figure 2 illustrates a basic UCM model consisting of a StartPoint (a path node) representing the triggering and the beginning of the execution of a scenario, a Path or scenario path that specifies the trajectory followed during the execution of the scenario, an arrow to show the direction of the execution, a Responsibility point representing a generic activity (e.g. action, task, operation, function), an EndPoint (a path node) that specifies the end of scenario execution or a point where the resulting output is accumulated (postcondition), and a UCM Team component that specifies a container. The UCM components describe the scenario structure and cover the environment and the architectural structure of a system. Components may contain other components. Paths, including any path node, may be superimposed over components, thus allocating scenario behavior to the scenario structure. As shown in Figure 3, different types of UCM components are used to describe various system artifacts (such as objects, processes, and software agents), as well as non-software entities (such as actors, hardware, etc.).

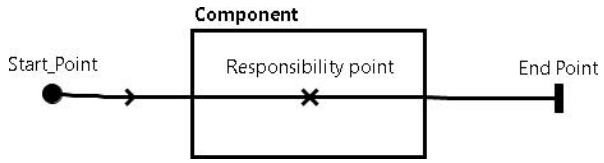

Fig. 2. Basic UCM elements.

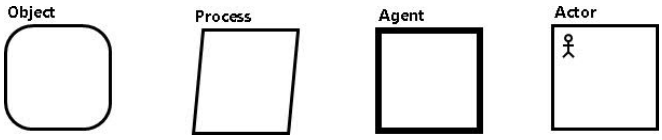

Fig. 3. UCM component elements.

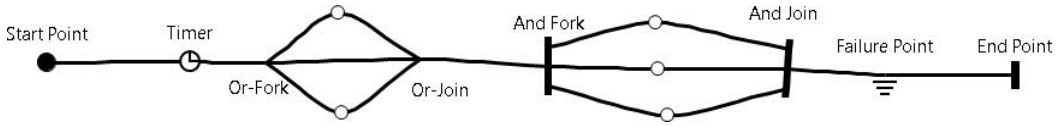

Fig. 4. UCM Path elements.

2) Other UCM Path Elements

Figure 4 presents other UCM Path elements that include Timer, OrFork, OrJoin, AndFork, AndJoin, and FailurePoint. A Timer is a path node and a special case of a waiting place, where the execution of the scenario relies on some conditions, such as the arrival of a trigger event or the occurrence of a timeout. An OrFork is a path node that splits a path segment into alternative branches in scenario execution. OrJoin is a path node that joins two or more incoming alternative path segments into a single and shared path in scenario behavior. AndFork is a path node that specifies parallelism by splitting an incoming path segment into two or more concurrent branches in scenario behavior, whereas, an AndJoin combines concurrent incoming branches into a single path. A FailurePoint is a path node that represents a point in scenario behavior where the continuation of the scenario depends on the occurrence of a failure or exception.

IV. THE PROPOSED APPROACH

A. Basic Idea

No single solution proposed for the achievement of an NFR is deterministic, and so, to complement the hard work done during the RE phase, the analysis of NFRs must continue throughout the entire software development process until the final product is generated. In general, with NFRs, the effectiveness of a solution becomes tangible when the system becomes operational or during the testing and validation phase. In addition, it is not possible to think about all possible alternatives during the RE phase or any other phase. Therefore, it is necessary to continuously analyze, develop, and manage NFRs throughout the software process in the same way that FRs are developed [24].

Beyond the RE phase, it is not always possible to distinguish between a system service described by FRs and a service that satisfies an NFR. Using the terminology proper to GORE methods, a solution or strategy for NFRs is generally composed of tasks and resources that are also used to model FRs. Consequently, from the security perspective, for instance, in an intermediate system model, both components describing system services may be equally vulnerable to those related to ensuring the security of the system. Therefore, further actions are needed to perpetuate the influence of security requirements on intermediate models until the final software product is obtained. The same applies to other NFRs, such as performance, usability, etc.

In summary, it would be necessary to have, in each software development phase, mechanisms to stimulate thinking about additional actions that may apply either to intermediate models or to the transformation process of such models to strengthen the satisfaction of NFRs.

B. Complementary Non-Functional Action (CNF-Action)

The CNF-Action specifies any activity (precisely an NF-Action) susceptible to contribute positively to the realization of an NFR (or its NF-Attribute) during the software design, development, and implementation phases. CNF-Action is meant to affect decision-making, modeling, and transformations throughout the SDLC phases. Examples of such actions include:

- Creating new functionality that may refine an existing solution or contribute directly to satisfy a Softgoal. In this case, its impact on other goals needs to be evaluated before its adoption.
- Guiding decision-making.
- Adding constraints to an activity.
- Transforming a model from one state to another.
- Ordering or restructuring some activities.

Some actions at one stage depend on precedent actions. For example, the design of parallel algorithms may induce the adoption of multicore systems at the implementation phase to achieve performance. The goal of the CNF-Action is also to continuously involve NFRs in decision-making and improve the quality of the intermediate models and the final software product. For the sake of traceability and continuity, CNF-Actions need to be documented and propagated or embedded in the models they affect.

C. CNF-Actions' Representation and Propagation

Thinking about how to represent CNF-Actions, the idea that first comes to mind is to progressively upgrade the requirements model (from the RE phase) whenever a new CNF-Action is derived. Considering the amount of work that may need to be repeated each time the model is upgraded (such as conflict and trade-off analysis, operationalization, re-evaluation of the entire model, etc.), it may be required to do so only when it is necessary (for example, when omitted requirements emerge at a later stage). In addition, if one model must grow continuously, it may end up becoming unmanageable. Thus, there is a need for an independent and more flexible approach.

Due to the various methods, techniques, and tools used throughout the software development process, it would be preferable to first consider the option to embed CNF-Actions in the models they have affected. Doing so would help to justify some aspects of the model during validation and facilitate backward traceability, and when the model is used as input, the knowledge of actions (or transformations) performed on the model is readily available. To this end, CNF-Actions can be integrated into a software model in various ways, e.g., they can be added to a UML model as profiles, or an additional class or

object. However, for portability and easy propagation, XML specification of CNF-Actions would be preferred in some circumstances. The use of XML could be further justified by the fact that XML languages are already intensively used in software engineering [25, 26]. Thus, although different methods and techniques may be used to model CNF-actions at different levels, for a given project, the following is a flexible template to which designers can add or remove components according to their needs:

$$< Phase, NFRs, CNF - Action, Domain, [optional] >$$

where $Phase$ is the SDLC phase, $NFRs$ are the list of NFRs affected by the action, the $CNF - Action$, $Domain$ contains the list of objects, components, and activities affected by the CNF-Action, and $optional$ is for any additional information.

D. A Non-Functional Requirement (NFR) Domain

Due to its generic nature, the concept of domain may have different meanings in different contexts, such as function domain in mathematics, domain name in computer networks, and application domain for software. In operating system design, the domain of protection is composed of hardware and software objects (see [27], Ch. 17, pp. 721-724). Since protection is an aspect of the security requirement that is indeed an NFR, this example is closely related to the context in which the term domain is used in this work. The CNF-Action domain can include software development activities, development methods and techniques, development tools, as well as the resources needed, intermediate software models, and the system environment.

A CNF-Action may constrain the selection of activities amongst alternatives, constrain an activity to be performed in a certain way, or include some specific functionalities. It may recommend or make mandatory the use of specific methods, techniques, and tools. E.g., a highly secure system may not be developed with unsecured tools. For portability, the Java language and tools may be recommended or even imposed for implementation, or a wired network (resource) may be preferred to wireless. With respect to the development and/or system environment, an operating system may be preferred over others to realize an NFR.

Due to the variety of entities that may be part of a domain as listed above, for the sake of the commodity and to facilitate operations on domains, the "object" concept can be used to abstract each domain element.

V. CASE STUDY

To illustrate the proposed CNF-Action concept, the case study of the electronic voting system in [28] was adopted. The software provides voters with the ability to vote in elections using a simple computer-based interface (electronic voting or e-voting). Its purpose is to make voting more accessible to the public, given the rapid proliferation of PCs around the world. From the comfort of their home, using the familiar Microsoft Windows interface, voters can securely access ballots and vote for their chosen representative. Four actors and two main softgoals are identified. The actors are voters, vote officers, a security company that ensures the safety of the physical resources needed for the system, and the voting system. The two softgoals are: the system should be highly secure and easy to use.

A. A GRL model of the case study

Since this work is concerned with the analysis of NFRs beyond the RE phase, the goal model proposed in [28] was reused, focusing on activities related to the design and implementation of the system. The GRL model of the case study is represented in Figure 8 of [28]. Based on the coloring of GRL elements resulting from its evaluation, the strategy adopted for the system includes the following:

- For system usability, users are provided with tips and audio while using the system, training videos are made available online, and graphical user interfaces should be used for user interaction to simplify the voting process. In addition, the use of biometric authentication further contributes to rendering the program easy to use.

For system security, biometric authentication is used to secure the software while the security of the server(s) and other physical equipment is assigned to a security company. Table I summarizes the tasks that must be implemented and their respective contribution toward the achievement of the system's goals. These tasks are believed to be enough to illustrate the proposed CNF-Action concept. For a complete system, many more functionalities, such as voter registration, voter verification and validation, vote counting, etc., should have to be included.

B. The Design Models of the Case Study

The architectural design of a system is important as it presents a holistic view and discloses its important parts/components, as well as the connections between them. The UCM of the system derived from the GRL model in Figure 8 [28] is represented in Figure 10 [28]. It is re-used here and progressively refined using CNF-Actions to illustrate the applicability of the concept.

TABLE I. SUMMARY OF TASKS TO SATISFY SYSTEM'S GOALS

Task	Description	Contribution
Audio tips	The system provides users with tips and audio to assist during the operation.	Provides instant assistance to users and facilitates its use.
Provide training videos	Training videos are directly accessible to users. Can be played online or downloaded.	Provides online/offline training to facilitate system use.
Casting a vote	Allows a voter to access the ballot and vote.	System service
Develop GUI	Use Windows forms decorated with appropriate drawings or pictures to guide a user throughout the voting process.	Improves the usability of the voting system and simplifies the voting process
Biometric authentication	Allows users with different backgrounds to easily access the system and be identified and authenticated.	Ensures software security.

TABLE II. BACKWARD TRACEABILITY FROM UCM TO GRL MODEL

No	UCM Elements	GRL Elements
1	Need Assistance, Generate tips and audio, Tips displayed	Tips with Audio
2	Need Training, Access training Video, Video watched	Provide training videos
3	Ready to vote, Voting process, Voted	Casting a vote
4	Voter biometric, Biometric authentication, Biometric accepted, Biometric denied	Biometric authentication
5	Voter (component)	Voter (actor)
6	Election officer (component)	Vote officer (actor)

As shown in Table II, the elements of the GRL model are linkable to those of the UCM, not only to validate the UCM model but also to establish backward traceability and ensure that all the elements of the input GRL were considered in the process of generating the UCM. If the UCM refinement process is not carefully conducted, it can become very challenging to trace the GRL elements back from those of the refined model.

Password authentication was excluded following the GRL model evaluation. The hardware security goal was deliberately excluded and, consequently, the security company actor and the service that it provides. This was to focus solely on software-related features.

C. Communication Over a Network/Internet Connection

The architectural design may stimulate thoughts on aspects of the system that were not thought of during the requirements phase. This focuses not only on "what" but also on "how", e.g., what components are needed for the system and how they are interconnected. Although the network/Internet communication could have been identified as a GRL resource needed by users to connect to the system for vote casting, download training video or play it online, it went undetected. At this point, it is elicited since the system architecture must answer questions such as how a user obtains a video from the system or how biometric data are transferred to the system. Figure 5 represents the UCM of the system with the network/Internet component integrated.

As shown in Figure 5, all communication between the voters and the system is via a network. It may be argued that the election officer should have direct access to the server that hosts the system. However, this does not exclude the possibility for all users to access the system via a network or an online connection. So, for illustration and clarity purposes, Figure 5 shows only the communication between the voter and the system that passes through the network. The communications between the election officer and the system are also meant to be via a network/Internet connection. To be authenticated, the biometric data of a voter is sent to the system through the network, and the system analyzes the data and sends feedback back to the voter through the same network. In the same vein, a request for a training video is sent to the system.

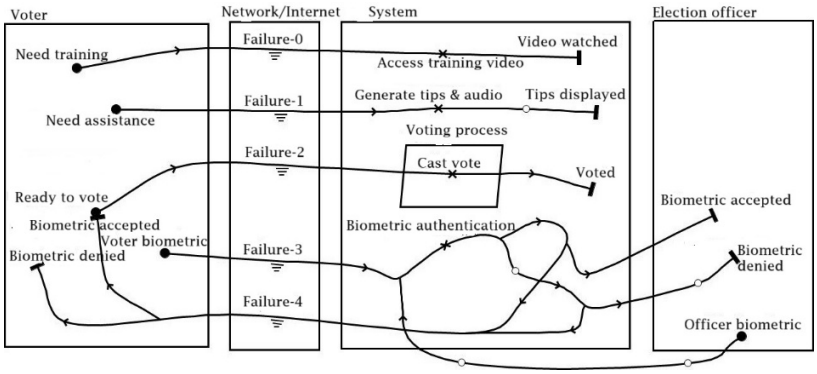

Fig. 5. UCM of the case study with the network component.

The integration of the network component into the system architecture is due to the ability of the design and modeling process to allow the elicitation of new or omitted requirements [28]. Such a requirement is not a CNF-Action, as it is not directly meant to reinforce the realization of an NFR. So, in practice, the GRL model must be updated with the newly elicited requirement, e.g., all users should access the system via a network/Internet connection. The updated model should also be re-evaluated since the new requirement may have a considerable impact on the initial strategy.

The failure point inside the network component in Figure 5 specifies the fact that communication over a network can fail and, hence, may constitute a threat to the usability of the system. It is also a threat to the security of the system since unauthorized users can access data transmitted over a network [29]. Thus, there is a need for CNF-Actions to address these issues.

D. Addressing Usability and Security Concerns

Having to wait forever in front of a computer or a mobile device for a response from the network or Internet does not contribute to facilitating the use of a system. Therefore, a mechanism is needed to detect communication failure and an approach to handle the failure. In Figure 6, the mechanism is represented with a Timer (a UCM object), a timeout path to handle failure, and an input and an output path segment. To further facilitate the use of the system, training videos (and, if possible, an MOC voting system) must be made accessible to the public. Thus, regarding the security of the system, the UCM

model can be divided into two main parts or subsystems: one that includes the functionalities related to biometric authentication and the voting process, namely the electoral subsystem, and the other subsystem containing the remaining functionalities accessible to the public, namely the training subsystem. Then, all communication within the electoral subsystem can be encrypted to enhance the satisfaction of the security requirement. This is illustrated in Figure 7.

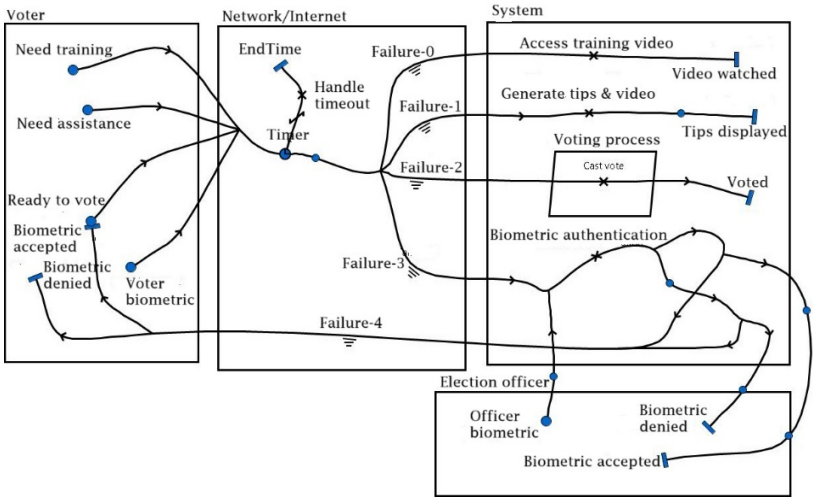

Fig. 6. UCM of the case study with a network control module.

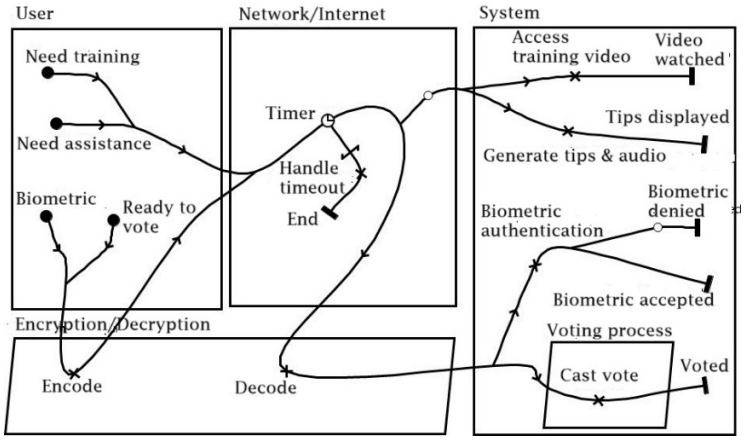

Fig. 7. UCM of the case study with encryption.

As shown in Figure 7, requests for training videos and tip and audio assistance are not encrypted, while the other two requests for biometric authentication and voting are encrypted before being sent to the system through the network/Internet connection. On the other hand, when the biometric data reaches the system, they are first decrypted before being authenticated. Similarly, when the system receives a request to cast a vote, it is first decrypted before being processed. Although not discussed in detail, an MOC system is a copy of the entire system for which security is not (fully) reinforced. For example, for biometric authentication, the same sample data can be used for any user who successfully completes the biometric capture process.

E. Highlights of the Contribution of CNF-Actions to the Case Study

The CNF-Actions performed on the UCM design of the case study are the following:

- The refinement of the UCM model with a timeout-recovery mechanism to control the transmission of data over the network/Internet to improve system usability.
- The elicitation of a new functionality to handle data transmission failure over the network.
- The division of the functionalities of the system into two groups (subsystems) to facilitate the reinforcement of the security measures on critical functionalities.

- The elicitation of new functionalities (encode and decode) to ensure the security of critical data exchanged between users and the system through the network.
- The refinement of the UCM model with a UCM process element to ensure the encryption and decryption of data transmitted over the network.

For the documentation and propagation of these CNF-Actions, the following structure is suggested but not compulsory, as the developer can decide otherwise. However, it is required to include the essential elements of each CNF-Action.

$$< Phase, NFR, CNF - Action, Domain, [optional\ ] >$$

The following data associated with the first CNF-Action are given for illustration:

Design Phase - UCM modeling, NFR: system usability, CNF-Action: Create network control mechanism: add a Timer object to the UCM, Or-Join all path segments that pass through the network to the system, Or-Fork path segments from network to the system, add a failure path to the Timer and a responsibility point to handle network failure. Domain: UCM model <network component>.

The data describing CNF-Actions can be specified using any approach, preferably XML, but, in the first place, it must be embedded in the UCM map as a text description or part of the model to be understandable to whoever uses it and justify its status. Having access to the CNF-Action data also helps ensure the backward traceability necessary to validate the UCM model. In addition, the XML specification of the data would facilitate the propagation of CNF-Actions to other software development phases, for instance, for better decision-making during the transformation of the UCM model.

VI. DISCUSSION AND FUTURE WORKS

This study introduced the concept of CNF-Actions to stimulate the analysis of NFRs at every stage of software development. This concept was used for the transformation of the UCM of the case study to ensure that the usability and security of the system are realized in the model. It was observed that CNF-Actions are not new software or system requirements, as they depend on objects they are meant to transform (domain). If a technique other than UCM was used for architectural design, the derived CNF-Actions would be different. However, the goal and reasoning do not change as far as the model or activity is concerned. That is, analyze the model or activity in the light of NFRs to derive the necessary actions that must be performed to ensure that the model or activity not only models the services of the system but also realizes the NFRs.

For instance, since the voting process must be simplified, it could be decided to specify it using the Z notation, with the benefit of exploiting its mathematical precision, to specify and optimize each detailed aspect of the specification. In that case, the CNF-Action concept would be applied to the formal specification in the same vein to ensure the usability and security of the system. Similarly, it can be decided to use UML to specify the user interfaces and other aspects of the system. In that case, the CNF-Action concept would be applied to UML in the same way it was applied to UCM.

The CNF-Action concept must be automated, preferably, as a plugin or component that can be integrated into existing software tools. In addition, an empirical study of this work must be carried out by applying it to industry-level projects.

VII. CONCLUSION

Previous studies on NFR analysis focused on the requirements phase [6], including, among others, NFR elicitation, classification, prioritization, specification, conflicts and tradeoff analysis, NFR operationalization, evaluation, and selection of the best strategies for satisfying NFRs. Despite calls from scholars for means or processes to continue the development of NFRs throughout the SDLC and in parallel with FRs, not much has been proposed to date. This study proposes the concept of CNF-Action to address this issue. The strength of the concept is its ability to continuously stimulate the analysis of NFRs throughout the software development process to ensure that any important decision, modeling, or activity does not focus solely on FRs but also considers NFRs. The concept was successfully applied to a case study to illustrate its applicability, improving the existing NFR development process by extending the process-oriented analysis beyond the RE phase.

REFERENCES

[1] I. Sommerville, *Software Engineering*, 10th ed. Boston, MA, USA: Pearson, 2015.

[2] S. Graf, Ø. Haugen, I. Ober, and B. Selic, "SVERTS – Specification and Validation of Real-Time and Embedded Systems," in *UML Modeling Languages and Applications*, 2005, pp. 33–42, https://doi.org/10.1007/978-3-540-31797-5_4.

[3] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, vol. 50, no. 9–10, pp. 833–859, Aug. 2008, https://doi.org/10.1016/j.infsof.2008.01.006.

[4] J. Spivey, *The Z Notation: A Reference Manual*, 1st ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1989.

[5] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Springer Science & Business Media, 2012.

[6] C. Dongmo, "A Review of Non-Functional Requirements Analysis Throughout the SDLC," *Computers*, vol. 13, no. 12, Nov. 2024, Art. no. 308, https://doi.org/10.3390/computers13120308.

[7] K. Y. Cai, "Non-Functional Computing: Towards a More Scientific Treatment to Non-Functional Requirements," in *31st Annual International Computer Software and Applications Conference - Vol. 2 - (COMPSAC 2007)*, Beijing, China, Jul. 2007, pp. 493–494, https://doi.org/10.1109/COMPSAC.2007.156.

[8] G. Grau and X. Franch, "A Goal-Oriented Approach for the Generation and Evaluation of Alternative Architectures," in *Software Architecture*, 2007, pp. 139–155, https://doi.org/10.1007/978-3-540-75132-8_12.

[9] L. M. Cysneiros, J. C. S. do Prado Leite, and J. de Melo Sabat Neto, "A Framework for Integrating Non-Functional Requirements into Conceptual Models," *Requirements Engineering*, vol. 6, no. 2, pp. 97–115, Jun. 2001, https://doi.org/10.1007/s007660170008.

[10] N. S. Rosa, P. R. F. Cunha, and G. R. R. Justo, "Process/sup NFL/: a language for describing non-functional properties," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, Big Island, HI, USA, 2002, pp. 3676–3685, https://doi.org/10.1109/HICSS.2002.994496.

[11] A. Almeida, N. Bencomo, T. Batista, E. Cavalcante, and F. Dantas, "Dynamic decision-making based on NFR for managing software variability and configuration selection," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, Dec. 2015, pp. 1376–1382, https://doi.org/10.1145/2695664.2695875.

[12] L. Chung and J. C. S. do Prado Leite, "On Non-Functional Requirements in Software Engineering," in *Conceptual Modeling: Foundations and Applications*, Springer, 2009, pp. 363–379.

[13] F. Dalpiaz, X. Franch, and J. Horkoff, "iStar 2.0 Language Guide." arXiv, Jun. 16, 2016, https://doi.org/10.48550/arXiv.1605.07767.

[14] M. Fatima, "KAOS: A Goal Oriented Requirement Engineering Approach," vol. 1, no. 10.

[15] "Z.151 (10/12) - User Requirements Notation (URN) - Language Definition." ITU-T, 2012.

[16] N. AlAmoudi, J. Hassine, and M. Baslyman, "GRLMerger: an automatic approach for integrating GRL models," *Requirements Engineering*, vol. 29, no. 2, pp. 209–259, Jun. 2024, https://doi.org/10.1007/s00766-024-00413-6.

[17] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, May 2004, https://doi.org/10.1023/B:AGNT.0000018806.20944.ef.

[18] S. Wolny, A. Mazak, C. Carpella, V. Geist, and M. Wimmer, "Thirteen years of SysML: a systematic mapping study," *Software and Systems Modeling*, vol. 19, no. 1, pp. 111–169, Jan. 2020, https://doi.org/10.1007/s10270-019-00735-y.

[19] F. G. C. Ribeiro, C. E. Pereira, A. Rettberg, and M. S. Soares, "Model-based requirements specification of real-time systems with UML, SysML and MARTE," *Software & Systems Modeling*, vol. 17, no. 1, pp. 343–361, Feb. 2018, https://doi.org/10.1007/s10270-016-0525-1.

[20] S. P. J. Abo'o Zo'o, "A framework for software quality assurance," Ph.D. dissertation, North-West University, South Africa, 2021.

[21] M. Saunders, P. Lewis, and A. Thornhill, *Research Methods for Business Students*. Pearson Education, 2009.

[22] D. Amyot, *Use Case Maps Quick Tutorial*. Canada: SITE, University of Ottawa, 1999.

[23] T. Binalialhag, J. Hassine, and D. Amyot, "Static slicing of Use Case Maps requirements models," *Software & Systems Modeling*, vol. 18, no. 4, pp. 2465–2505, Aug. 2019, https://doi.org/10.1007/s10270-018-0680-7.

[24] C. Werner, Z. S. Li, D. Lowlind, O. Elazhary, N. Ernst, and D. Damian, "Continuously Managing NFRs: Opportunities and Challenges in Practice," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2629–2642, Jul. 2022, https://doi.org/10.1109/TSE.2021.3066330.

[25] Y. Fei and Z. Xiaodong, "An XML-Based Software Non-Functional Requirements Modeling Method," in *2007 8th International Conference on Electronic Measurement and Instruments*, Xian, China, Aug. 2007.

[26] P. M. S. Poon, T. S. Dillon, and E. Chang, "Transformation of QoS data into XML characterising data communication in real time distributed systems," in *2nd IEEE International Conference on Industrial Informatics, 2004. INDIN '04. 2004*, Berlin, Germany, 2004, pp. 204–209, https://doi.org/10.1109/INDIN.2004.1417330.

[27] A. Silberschatz, G. Gagne, and P. B. Galvin, *Operating System Concepts*, 10th ed. Wiley, 2018.

[28] C. Dongmo and J. A. Van Der Poll, "An Improved User Requirements Notation (URN) Models' Construction Approach," *Systems*, vol. 11, no. 6, Jun. 2023, Art. no. 301, https://doi.org/10.3390/systems11060301.

[29] M. Tarhda, R. E. Gouri, and L. Hlou, "Implementation of an Optimized Steganography Technique over TCP/IP and Tests Against Well-Known Security Equipment," *Engineering, Technology & Applied Science Research*, vol. 8, no. 6, pp. 3515–3520, Dec. 2018, https://doi.org/10.48084/etasr.2334.