Non-Functional Requirements Discovery and Quality Assurance using Goal Model for Earthquake Warning System in Operation

Youngsul Shin
Software Disaster Research Center
Kyungpook National University
Daegu, Republic of Korea
youngsulshin@gmail.com

Seok-Won Lee*
Dept. of Software & Computer Engr.
Dept. of Artificial Intelligence
Ajou University
Suwon, Republic of Korea
leesw@ajou.ac.kr

Yunja Choi*
School of Computer Science & Engr.
Kyungpook National University
Daegu, Republic of Korea
yuchoi76@knu.ac.kr

***Abstract*—Many industrial systems that are developed without proper engineering guidance due to a lack of expertise or resources suffer from failures and maintenance problems in their evolving lifecycle. For mission-critical systems, in particular, ensuring high quality of non-functional requirements in a rapidly changing domain environment is of the utmost importance. In this paper, we report our case study with an industry system, a sensor-based earthquake warning system that was developed without a rigorous engineering process. Therefore, no requirements documents are available for future maintenance and verification. In this study, we used various types of software analysis methods such as stakeholder interviews, document reviews, source code analysis, model checking, and software testing for discovering requirements and also for verification purposes. We used a goal modeling approach to gather a set of initial requirements as though they had been elicited using an appropriate requirements engineering method in the early stage of the development process. Furthermore, software testing and model checking were iteratively used to verify and clarify unknown-source, uncertain, and unconfirmed requirements during the revision of the goal model. This study also provides an architectural improvement of the system through the discovery of requirements conflicts and violations. The experience and findings of this study, which demonstrate the effectiveness of applying diverse software engineering techniques in maintenance, can contribute to the analysis and evolution of systems developed without a proper engineering process, by discovering and verifying some critical requirements specifications.**

***Index Terms*—requirement discovery, requirement verification, goal model, quality assurance, maintenance and evolution**

I. Introduction

Requirements discovery is one of the most important and critical activities in the development of software systems, especially when the system is a complex, mission-critical, and distributed system expected to be maintained for a long period of time [1]. A high-quality system can be achieved by applying software engineering principles incrementally and iteratively, evolving systems from well-defined requirements to high-level architectural design, low-level detailed design, and finally to source code implementation. This way makes the system more maintainable, with highly modular and coherent software artifacts that can be easily traced from requirements to source code. In contrast, failure to discover important functional/non-functional requirements up front is highly likely to result in low-quality systems being developed that are error-prone and costly to maintain.

Unfortunately, it is rare to find systems in the industrial field that properly apply software engineering principles [2], mainly due to lack of project time, lack of funds, lack of leadership who are aware of the importance of applying software engineering principles, and lack of engineers trained well with those principles. The situation is similar even in the development of mission-critical social infrastructure systems, such as medical systems, transportation systems, and safety management systems.

The Earthquake Warning System (EWS) is one such system developed and being operated without the application of software engineering principles. The EWS utilizes acceleration data transmitted by sensors installed nationwide to detect actual seismic waves and determine the occurrence of earthquakes. It is a mission-critical social safety infrastructure system composed of numerous distributed components. It was developed without properly performing fundamental software engineering activities, such as requirements discovery and analysis, architectural design, structural/behavioral design, and systematic testing. The absence of requirements documents, a design for non-functional requirements, and a systematic verification process are problems EWS has been suffering from starting from the early stage of its development. As a result, it has been extremely difficult to identify the source of problems that have been witnessed during its years of operation. Throughout the operation of the EWS, developers have continuously observed the phenomenon of message sequence inversion, where the temporal order of messages sent by sensors is reversed. Developers speculated that this phenomenon might be an error, and continued to operate the EWS without finding evidence of exactly what impact this phenomenon has or what solutions exist for this phenomenon.

*Corresponding authors.

This study presents our experience with re-engineering the EWS from requirements discovery to high-level architecture design by utilizing all available software artifacts, resources, and verification techniques that might be helpful, including interviewing developers, analyzing available documents and program source code, and applying testing and model checking techniques. At the core of our re-engineering approach lies the construction of a goal model—a hierarchical representation of requirements where high-level goals are broken down into subgoals, illustrating hierarchical relationships among goals [3].

Our approach consists of three phases: initial requirements discovery, goal relationship discovery, and goal model verification and requirements discovery. The initial requirements discovery phase identifies initial requirements through developer interviews, document reviews, and source code analysis. The goal relationship discovery phase constructs a goal model by finding relationships between goals using the discovered requirements. In the goal model verification and requirements discovery phase, software testing and model checking techniques are used to verify this goal model, iteratively evolving the initial goal model into a refined goal model by identifying new requirements.

During the goal model construction process, we were able to identify critical performance requirements that had been under suspicion but remained unclear and uncertain. By testing the EWS w.r.t the finally constructed goal model, we were able to identify specific goals related to non-functional requirements that the EWS failed to satisfy and proposed an architectural solution to improve the structure and performance of the EWS to satisfy these goals. Our experience demonstrates the importance of having explicitly stated and well-structured requirements for continuous quality improvements and maintenance. In summary:

- We present a process for constructing a goal model of AI[1]-enabled, large-scale distributed, and mission-critical systems that have no requirements specifications.
- We discovered requirements using interviews, document review, and source code analysis. We also verified and evolved the goal model using software testing and model checking.
- We used the goal model to identify the issues in the EWS and present an architectural solution that satisfies the goal model.
- This study demonstrates that goal models, constructed with the help of various existing software engineering techniques, serve as an effective tool for managing the minimal essential requirements necessary for system maintenance and evolution.

The remainder of this paper is organized as follows. Section II describes the EWS, and Section III presents the process for constructing a goal model of the EWS, which was originally developed without requirements specifications. Section IV shows the improvement in the quality of the EWS achieved by addressing the issues identified through the goal model. Section V reviews existing related work. In Section VI, we discuss our study findings and provide an outlook on future directions with lessons learned.

[1]Artificial Intelligence

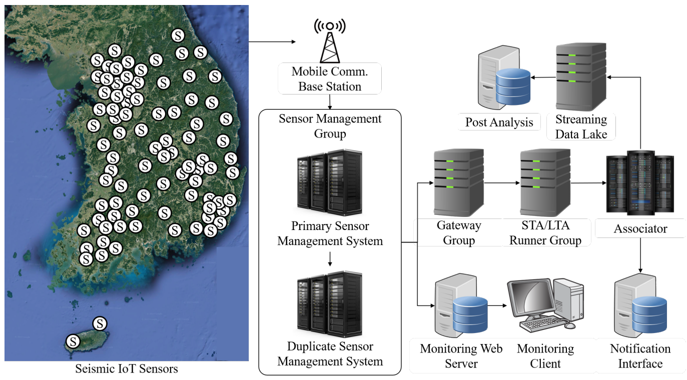


Fig. 1 Overview of the EWS.

II. Earthquake Warning System

The EWS is being developed by applying the findings of earthquake early warning research [4]. A wireless telecommunications operator in the Republic of Korea implements seismic IoT sensors and provides wireless telecommunication resources. The national meteorological observation agency of the Republic of Korea operates the EWS alongside the existing earthquake detection system to enhance its earthquake detection ability. The EWS has been in operation for more than five years, with 5,000 sensors distributed across the country.

A. Earthquake Inference Mechanism

An overview of the EWS is shown in Fig. 1. Acceleration data from sensors installed nationwide is transmitted to servers via mobile networks. The Sensor Management Group and the Gateway Group are clusters of servers that relay acceleration data to the STA/LTA Runner Group. Each STA/LTA Runner executes the short-term average/long-term average (STA/LTA) algorithm [5] to extract seismic signals from the acceleration data and forward them to the Associator. The Associator then uses the seismic signals to perform an AI inference mechanism that ultimately determines the occurrence of an earthquake.

The high-level architecture of the distributed components comprising the EWS is shown in Fig. 2. When a sensor connects to the Sensor Management System (SMS), a Channel is created. Data sent by the sensor goes to a Channel Queue mapped one-to-one to each Channel through the Handler. Once a complete message is merged in the Channel Queue, the message is passed through the Handler to the Global Queue. Ten Transmission Threads pull messages from the Global Queue and send them to their connected Gateway Handler. Subsequently, the message is delivered to the STA/LTA Runner, which analyzes the messages to extract seismic signals.

The Associator divides the observed area into hexagonal cells, each with side lengths of 9 kilometers. Within each cell, the number of sensors varies from 1 to 7, and the inter-sensor

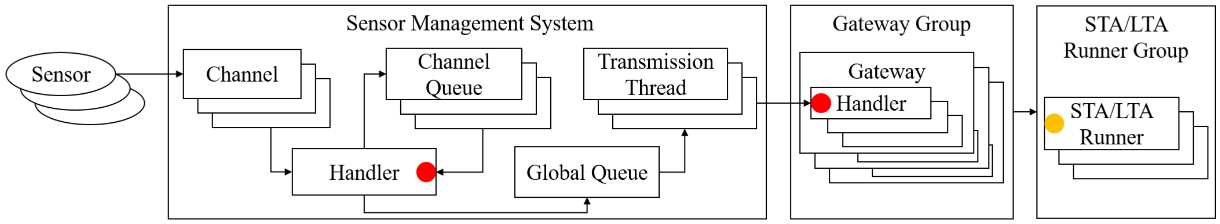


Fig. 2 Architecture of the EWS components.

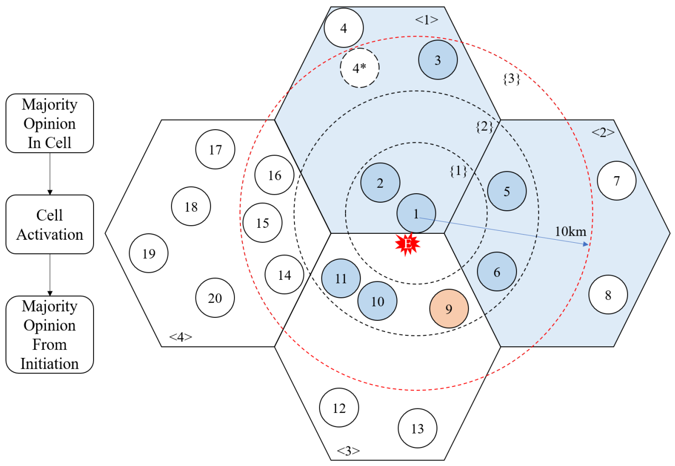


Fig. 3 Example of the earthquake inference mechanism.

distances are not uniform. The earthquake decision mechanism is based on a majority rule and proceeds as follows:

1) The sensor that first detects seismic signals (i.e., earthquake-like signals, but not determined as an earthquake yet) becomes the initial sensor.
2) If half of the sensors or more within a given cell detect seismic signals, the cell becomes activated.
3) If the threshold value $T{=}D/N$ is 0.5 or greater, the EWS concludes that an earthquake has occurred, where
   - D is the number of sensors that extracted seismic signals within 2 seconds of the initial sensor detecting signals; those sensors are found both inside the active cell and within 10 kilometers of the location of the initial sensor,
   - N denotes the total number of sensors situated within 10 kilometers of the initial sensor's location.

Fig. 3 illustrates an example of the earthquake inference mechanism. Assume the red icon labeled 'E' is the earthquake epicenter and sensor 1 in cell $\langle 1 \rangle$ first detects the seismic signal. When the seismic signal propagates to location $\{1\}$, sensor 2 detects it and, since two of the four sensors in cell $\langle 1 \rangle$ detect the seismic signal, cell $\langle 1 \rangle$ becomes activated. When the seismic signal propagates to location $\{2\}$, sensors 5, 6, 9, 10, and 11 also detect the signal, and cells $\langle 2 \rangle$ and $\langle 3 \rangle$ become activated. Sensor 4* is a hypothetical replacement for sensor 4, which will be explained later in this paper. A total of 11 sensors are located within a 10 km radius from the initial sensor 1, marked by the red dotted circle. Of those, seven sensors qualify for majority voting because (1) they are located in an activated cell, (2) they detected the earthquake within 2 seconds, and (3) they are located within the 10 km radius from the initial sensor that detected the seismic signal. Since a majority, i.e., seven of the 11 sensors, claim to have detected the earthquake, the system concludes that an earthquake has occurred and sends out alarms.

B. Development and Maintenance Issues

Despite being a mission-critical and AI-enabled system, the EWS lacked the application of software engineering practices in its design and implementation to a significant extent, which ultimately led to system maintenance failures. The software engineering issues present during the development phase of the EWS include:

- There are no artifacts such as explicit requirements documentations and structural designs for the EWS.
- The development focused primarily on functional implementation, with no analysis of non-functional requirements from a mission-critical system perspective.
- Design considerations for non-functional requirements were not taken into account.
- The EWS was developed by more than three companies, and due to the relationships among some of the stakeholders, certain artifacts, including source code, could not be provided for system analysis and verification. This has a very negative impact on system quality.
- There was no systematic verification process during the development and operation phases of the EWS. It especially lacks methods and activities for non-functional testing.
- There is no system in place for the handover of duties in case a developer is assigned to other tasks.

Due to these software engineering issues, it was inevitable that the developers would not understand non-functional requirements and system maintenance failures. They focused solely on quickly transmitting the messages produced by the sensors to the STA/LTA Runner and unconsciously assumed that the messages would arrive in the order they were sent. There was no analysis of the time constraints for transmitting sensor messages, and no verification was performed to confirm the assumption that messages would arrive in order.

The phenomenon of message sequence inversion, where messages sent from each sensor do not arrive at the STA/LTA Runner in chronological order, was continuously observed throughout the operation of the EWS. Developers were aware of this issue and suspected that it could be a source of critical failures, but they lacked concrete evidence. As they could understand only part of the system they were developing, the only possible solution they could apply was to filter out wrong message sequences at the programming level. The impact of this patchwork on the entire system has never been analyzed.

III. Goal Model Construction

Our goal model construction process, which is shown in Fig. 4, consists of three phases: initial requirements discovery, goal relationship discovery, and goal model verification and requirements discovery.

The initial requirements discovery phase, which includes activities such as developer interviews, document reviews, and source code analysis, identifies unknown requirements to construct an initial goal model. The goal relationship discovery phase defines goals and relationships among the goals to build a goal model. An initial goal model is constructed from the initial requirements discovered from the initial requirements discovery phase by first determining the root goal and recursively refining each goal into sub-goals until all discovered requirements are included in the initial goal model. We note that this initial goal model is neither complete nor certain. The goal model verification and requirements discovery phase verifies uncertain goals w.r.t their clarity, validity, and potential conflict with their parent goals, using software testing and model checking techniques. Unclear, conflicting, or invalid goals identified in this phase help us to discover additional new requirements, leading to a new goal model. These two phases are iterated until each uncertain goal in the goal model is further clarified.

Table I presents the requirements identified through the two phases of discovery for goal model construction: the initial requirements discovery phase and the goal model verification and requirements discovery phase. Figure 7 shows a part of the goal model we constructed through iterations of requirements discovery and goal model verification.

A. Initial Requirements Discovery

1) Interview: Interviews enabled us to infer requirements not documented in the form of documents or source code. In Table I, requirements identified by IDs beginning with INT and the REQ-UCT requirements were discovered during the interview process. As a non-functional requirement, INT1-US is typically decided after reviewing the integration performance of the message processing system; however, the basis for determining whether providing an alarm within 4 seconds is a correct time constraint for earthquake evacuation still remains unclear, making it a requirement of unknown-source status.

Other requirements such as INT4-UV, INT5-UV, INT6-UV, INT7-UV could be identified if the source code were available. Although the sensor nodes and the AI-enabled earthquake inference algorithm are distributed components of the EWS, we were not provided with the source code of these components due to conflicting relationships among the stakeholders. Therefore, these four requirements could only be discovered through interviews, and since no source code was provided, they are in an unverifiable status.

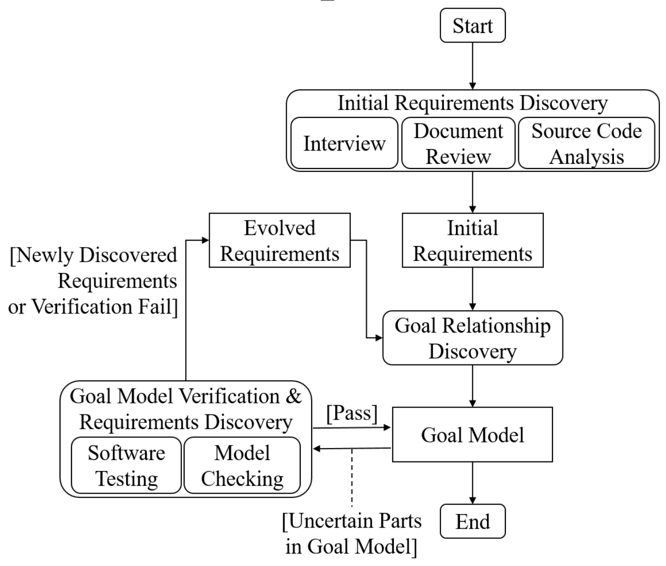

Start
Initial Requirements Discovery
Interview
Document Review
Source Code Analysis
Evolved Requirements
Initial Requirements
[Newly Discovered Requirements or Verification Fail]
Goal Relationship Discovery
Goal Model Verification & Requirements Discovery
Software Testing
Model Checking
[Pass]
Goal Model
[Uncertain Parts in Goal Model]
End

Fig. 4 Process for constructing a goal model.

Requirement REQ-UCT is of uncertain status. We received an answer from the developers that they were not certain whether it was a requirement. The developer who implemented the component assumed that messages sent first on the network are received first as a matter of course and considered REQ-UCT as an obvious requirement. However, the developer did not explicitly verify the necessity of this requirement. In fact, the formation of developers' unconscious assumptions is a phenomenon that occasionally occurs in industrial practice due to their familiarity with the domain.

2) Document Review: The requirements that can be extracted vary depending on the type of document. The document given to us, entitled 'EWS System Introduction Document', provided a high-level overview of the technology required for EWS implementation. By reviewing this document, we were able to extract both functional and non-functional requirements.

In Table 1, requirements identified by IDs beginning with DOC were discovered during the document review process. DOC2 is a requirement that reflects the stakeholder's economic perspective of utilizing relatively inexpensive micro-electromechanical systems-based hardware to satisfy DOC0. The source of requirement INT1-US, which was in unknown-source status and discovered during the interview process, was found in the research by Fujinawa et al. [6]. Therefore, requirement INT1-US was renamed to DOC1.

3) Source Code Analysis: Source code analysis can identify technical requirements that cannot be extracted from interviews and document reviews. We were able to identify requirements related to the security aspects of the system and

TABLE I Discovered requirements.

ID	Status	Context
INT0	CFD	Establish a nationwide earthquake observation network
INT1-US→DOC1	US→CFD	Provide earthquake alarms to the entire population within 4 seconds of an earthquake occurrence
INT2	CFD	Ensure there are no false alarms in earthquake detection
INT3	CFD	Determine the occurrence of an earthquake within 2 seconds of its onset
INT4-UV	UV	Use seismic signals within a 10 km range from the sensor that first detects the seismic signal
INT5-UV	UV	Use seismic signals within 2 seconds from the time the earthquake signal is first detected
INT6-UV	UV	Send 100 acceleration data points every second as a single message to the server from the sensors
INT7-UV	UV	Sample acceleration every 10 milliseconds from the sensors
DOC0	CFD	Detect all earthquakes above a magnitude of 2.0
DOC2	CFD	Install MEMS-based sensors nationwide
SRC0	CFD	Receive only messages sent by pre-registered sensors
SRC1	CFD	Use interfaces for the transmission of acceleration data
SRC2	CFD	Use an acceleration message frame in the sensors
SRC3	CFD	Use an integrated type message frame on the servers
REQ-UCT→TST0	UCT→CFD	Receive sensor messages in chronological order by the seismic signal extraction component for each sensor
MC0	CFD	Relay as many messages as the number of sensors per second to prevent message delays on the servers

[*] CFD:Confirmed, US:Unknown Source, UV:Unverifiable, UCT:Uncertain.

even interface requirements for components that the source code did not provide. The requirements extracted by analyzing the EWS's source code are as follows:

SRC0 is a security requirement for false alarms in earthquake detection. SRC1, SRC2, and SRC3 are interface requirements between distributed components. Despite not having the sensor's source code, we were able to identify the sensor's interface requirement SRC2 by analyzing the interfaces of components that maintain interface relationships with the sensor. Requirement REQ-UCT, discovered during the interview process, was also identified during the source code analysis. However, even with the provided source code, it was not possible to conclusively determine that REQ-UCT is a certain requirement. Therefore, it remained an uncertain requirement.

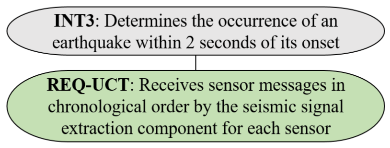
B. Goal Relationship Discovery

This phase identifies the relationships among the elicited requirements and produces a goal model. To find the goal relationships, we applied the method for constructing a goal model based on a flow of questions researched by Nakagawa et al. [7].

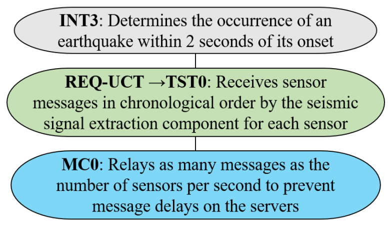
This method defines a set of questions that facilitate the discovery of the root goal, the identification of a subgoal's parent goal, and the aggregation of goals. The key questions are as follows:

- 'What is the purpose of the system?' (for root goal discovery)
- 'Is goal X a subgoal of goal Y?' (for identifying a subgoal's parent goal)
- 'Can these goals be aggregated?' (for goal aggregation)

The requirements analyst iteratively answers the flow of these questions to produce the goal model. While Nakagawa et al. introduced a tool for semi-automating goal model creation by interactively querying the analyst, we opted for a manual approach, creating the goal model by having one team member pose questions directly to the rest of the requirements analysis team.


INT3: Determines the occurrence of an earthquake within 2 seconds of its onset
REQ-UCT: Receives sensor messages in chronological order by the seismic signal extraction component for each sensor

(a) Part with the uncertain goal.


INT3: Determines the occurrence of an earthquake within 2 seconds of its onset
REQ-UCT →TST0: Receives sensor messages in chronological order by the seismic signal extraction component for each sensor
MC0: Relays as many messages as the number of sensors per second to prevent message delays on the servers

(b) Part with the derived goal.

Fig. 5 Part of the goal model with uncertain/derived goals.: (a) part with the uncertain goal, (b) part with the derived goal.

Based on the interviews and the source code analysis conducted before, we included the commonly identified uncertain requirement REQ-UCT in our initial goal model construction. Through careful consideration, we hypothesized that the parent goal of these uncertain requirements is INT3. Part of the goal model that includes this uncertain goal is shown in Fig. 5a. A goal model that includes uncertain goals must undergo the goal model verification phase to validate the hypotheses.

C. Goal Model Verification and Requirement Discovery

This phase validates the hypotheses established during goal model construction with uncertain goals. Proving the hypotheses true implies that:

- The uncertain goal is a valid goal.
- The relationship between the uncertain goal and its parent goal is true.

The results from software testing showed that the hypothesis related to the uncertain goal is true. Also, a new non-functional requirement was derived through software testing and its context was precisely defined by model checking. If the hypothesis had been false, the uncertain goal would have needed to be removed, necessitating reconstruction of part of the goal model.

1) Software Testing: We used software testing to prove the hypothesis that the EWS must satisfy requirement REQ-UCT in order to meet requirement INT3. To prove the hypothesis, it is sufficient to demonstrate that the EWS fails to satisfy INT3 if it does not meet REQ-UCT. The validation experiment using software testing proceeded in the following order:

1) Real seismic waves were represented using the acceleration message frames used by the sensors.
2) The message at the moment the seismic wave started was swapped with the message located one second later. This indicates that a message sequence inversion has occurred within a single message.
3) A message sequence with one message order inversion was input into the STA/LTA Runner, a component for extracting seismic signals for each sensor.
4) It was checked whether the STA/LTA Runner failed to extract seismic signals or if there was a delay in the extraction time.

Fig. 6 shows the experimental proof of the REQ-UCT hypothesis using software testing. As shown in Fig. 6a, the STA/LTA Runner component alone was isolated from the entire system and connected to a test driver. The test driver needed to simulate the behavior of the sensors and the message relay servers, so it was implemented to satisfy INT6-UV and SRC1 in the goal model shown in Fig. 7. To replicate the actual deployment of the large-scale distributed EWS with a small number of hosts, we set up the test environment using a virtual environment. We used test cases transformed from original seismic signals that actually occurred. The original signal indicates that the earthquake wave started at time *t0* and finished at time *t4*. Fig. 6b shows the test results when the test driver inputs the sampled messages from the original signal into the STA/LTA Runner in chronological order. The seismic signal extracted by the STA/LTA Runner was identical to the original signal. Fig. 6c shows a message sequence inversion. Since real sensors sample the seismic signal one second before transmitting it in message format, the portion of the signal from the start of the earthquake at time *t0* to *t1* is stored in the *M1* message. The subsequent parts of the signal, sampled at one-second intervals, are stored in messages following *M1*. The test driver feeds test cases that place message *M1*, which contains the beginning portion of the seismic signal, immediately after message *M2*. This test case exhibits a single instance of message sequence inversion. The signal extracted by the STA/LTA Runner started at *t3* and ended at *t4*, deviating from the original signal. Due to message sequence inversion, the STA/LTA Runner failed to extract the parts of the signal from *t0* to *t3*. This means that there was a 3-second delay in detecting the seismic signal.

We repeated the above experiment using 1,172 actual seismic signals obtained from the operator team. Among the 1,172 seismic signals, two cases resulted in total detection failure due to message sequence inversion. Additionally, there was one case of a 4-second detection delay, one of a 3-second delay, and seven of a 2-second delay. The remaining 1,161 cases experienced a 1-second delay.

The experimental results were analyzed in connection with the AI-enabled earthquake inference mechanism, as shown in Fig. 3, represented by INT3. Our analysis revealed that, if the messages from sensor 9 in Fig. 3 are not delivered in chronological order, causing the STA/LTA Runner to report to the Associator that sensor 9 did not extract the seismic signal, only sensors 1, 2, 3, 5, and 6 within the activated cells $\langle 1 \rangle$ and $\langle 2 \rangle$ will participate in the final decision. As only five of the 11 sensors extract the seismic signal, the Associator concludes that no earthquake has occurred. If sensor 4 happens to be located at point 4*, the Associator can recognize that an earthquake is occurring since six of the 12 sensors extracted the seismic signal; however, a time delay is introduced as the seismic signal needs to propagate further from location $\{2\}$ to location $\{3\}$.

The software testing results demonstrated that the EWS must satisfy requirement REQ-UCT in order to meet requirement INT3. Therefore, we can conclude that REQ-UCT has been correctly discovered and must be an important non-functional goal. REQ-UCT was renamed to TST0 by software testing, and its status was changed from uncertain to confirmed.

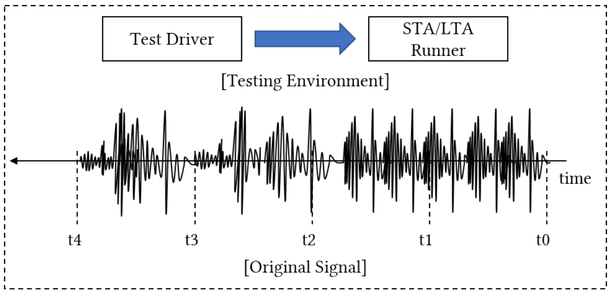
On the other hand, the violation case of TST0 showed that we needed to refine it to ensure the satisfaction of TST0.

2) Model Checking: From TST0 and testing, we found a reason to refine it in order to prevent message delays. Our intention was to find unknown necessary conditions (causes) of the delay so that we can assure the satisfaction of the requirements by preventing those conditions. We formulated our intention as a property:

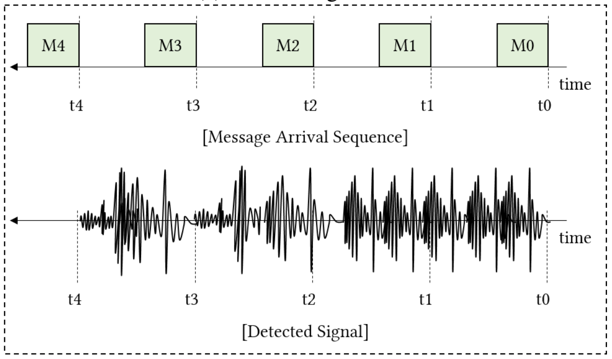
- Refinement property for TST0: Given a 1-second time limit and N sensors, the delivery of M messages from one server to another within the time limit effectively ensures the absence of any message sequence inversion.

By varying the numbers N and M, we iterated model checking [8] to identify the minimum value M for a given N using exhaustive search strategies over possible sets of execution contexts. Given a model and a property expected to be satisfied by the model, model checking exhaustively searches through possible behaviors of the model in order to either prove that the model satisfies the property or refute this with counterexamples showing that the property can be violated. We modeled the EWS architecture using the input language Promela of SPIN [9], [10] and tried to verify the property by varying the values of N and M. Through this process, we were able to verify that M must be equal to N in order to prevent message sequence inversion, thereby discovering a new non-functional requirement MC0.

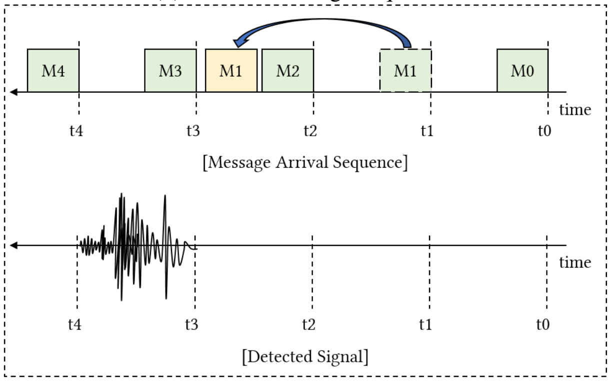
The part of the goal model shown in Fig. 5a evolved into the part of the goal model shown in Fig. 5b through the goal model verification and requirements discovery process. The



(a) Test configuration.



(b) Normal message sequence.



(c) Message sequence inversion.

Fig. 6 Impact of message sequence inversion: (a) testing configuration, (b) normal message sequence case, (c) message sequence inversion case.

final goal model of the EWS, constructed through the process in Fig. 4, is shown in Fig. 7.

IV. Quality Improvement using the Goal Model

Once the goal model had been constructed using the suggested iterative goal construction process, we used it to verify the EWS w.r.t. its functional/non-functional requirements. This section explains how the goal model-based requirements were used to identify issues in the EWS and how we improved the quality of the system by providing architectural solutions, which is similar to the activities of the "Twin Peaks" model [11] in software engineering.

A. Verification of the EWS w.r.t the Goal Model

We performed the verification of the EWS w.r.t. the goal model in a bottom-up manner, starting from leaf goals up to the root goal; For example, as shown in Fig. 7. In order to verify TST0, we verified SRC2 and SRC3, which imply SRC1, verified INT7-UV which implies INT6-UV, and then verified SRC1 and MC0, which imply TST0 under the condition that INT6-UV is true:

$$\begin{aligned} \text{INT6-UV} \wedge \text{SRC1} \wedge \text{MC0} &\rightarrow \text{TST0} \\ \text{INT7-UV} &\rightarrow \text{INT6-UV} \\ \text{SRC2} \wedge \text{SRC3} &\rightarrow \text{SRC1} \end{aligned}$$

As the engineers have long been suspecting the message inversion issue in the system, our first verification goal was MC0, one of the subgoals of TST0. We conducted performance testing to verify that the EWS satisfies MC0 by transmitting 5,000 messages per second from the 5,000 simulated sensors to the SMS and by checking transmission time between the SMS and the Gateways. The result is shown in the box-and-whisker plot in Fig. 8, where the y-axis displays the time needed to transmit 5,000 messages in milliseconds. The median performance of the SMS and the Gateway was 1,676 milliseconds and 329 milliseconds, respectively. The interquartile range was 222.2 and 20, respectively. The Gateway satisfied MC0, but SMS failed to meet MC0. We found that the cause of the message sequence inversion in the EWS was due to the low performance of the SMS.

For the goals INT6-UV and INT7-UV, for which source code or executable code was not provided because they were developed by third-party vendors, we confirmed that the sensors' performance requirements were satisfied through third-party testing. We also confirmed that the EWS satisfies SRC1, SRC2, and SRC3 through source code analysis. Therefore, we only needed to resolve the violation of MC0 to meet TST0.

B. Proposed Architectural Solution

After identifying the source of the issue as the low performance of the SMS, we tried several approaches to improve its performance, such as (1) increasing resources and adjusting configuration parameters, (2) carrying out a source code analysis to identify the fundamental reason for the low performance, and (3) developing architectural solutions to address the issues identified by the source code analysis.

The first approach yielded no performance improvements. We identified the fundamental reason of the low performance through source code analysis as transmission delays from the SMS to the Gateway, where each transmission thread waits for the result of the network transmission. Therefore, we suspected that architectural refactoring could be the only solution and suggested a possible architectural change: the use of a fully asynchronous multi-threaded message transmission mechanism using the Netty framework [12].

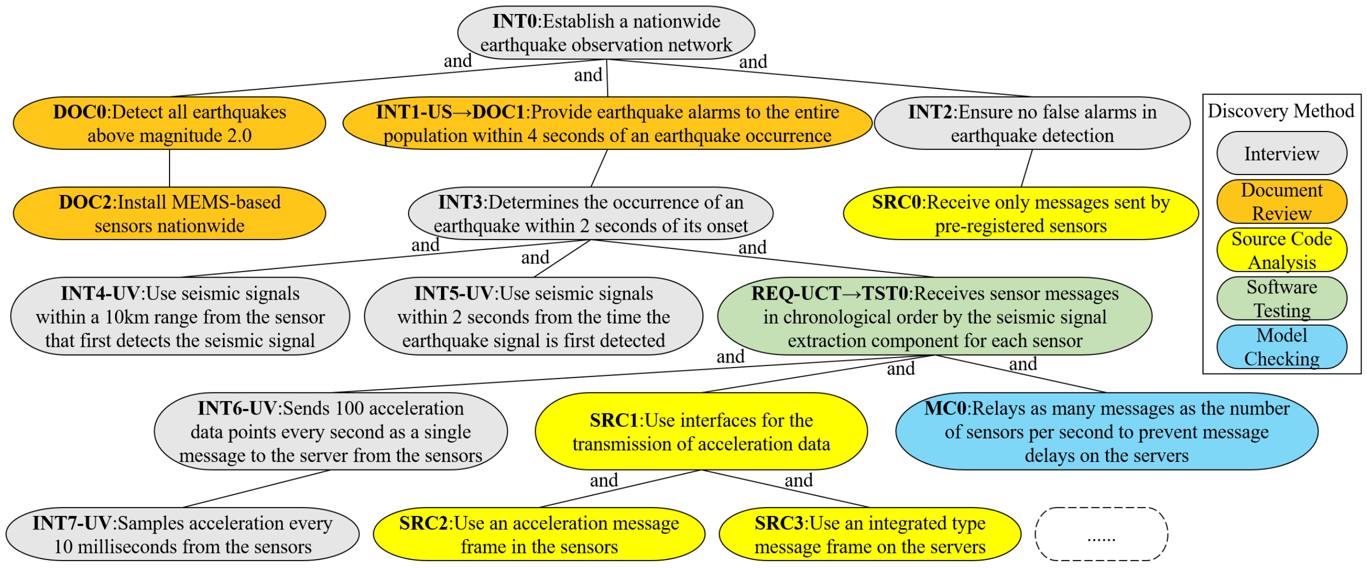

INT0:Establish a nationwide earthquake observation network
and
and
and
DOC0:Detect all earthquakes above magnitude 2.0
INT1-US→DOC1:Provide earthquake alarms to the entire population within 4 seconds of an earthquake occurrence
INT2:Ensure no false alarms in earthquake detection
DOC2:Install MEMS-based sensors nationwide
INT3:Determines the occurrence of an earthquake within 2 seconds of its onset
SRC0:Receive only messages sent by pre-registered sensors
INT4-UV:Use seismic signals within a 10km range from the sensor that first detects the seismic signal
INT5-UV:Use seismic signals within 2 seconds from the time the earthquake signal is first detected
REQ-UCT→TST0:Receives sensor messages in chronological order by the seismic signal extraction component for each sensor
INT6-UV:Sends 100 acceleration data points every second as a single message to the server from the sensors
SRC1:Use interfaces for the transmission of acceleration data
MC0:Relays as many messages as the number of sensors per second to prevent message delays on the servers
INT7-UV:Samples acceleration every 10 milliseconds from the sensors
SRC2:Use an acceleration message frame in the sensors
SRC3:Use an integrated type message frame on the servers
......
Discovery Method
Interview
Document Review
Source Code Analysis
Software Testing
Model Checking

Fig. 7 Constructed goal model.

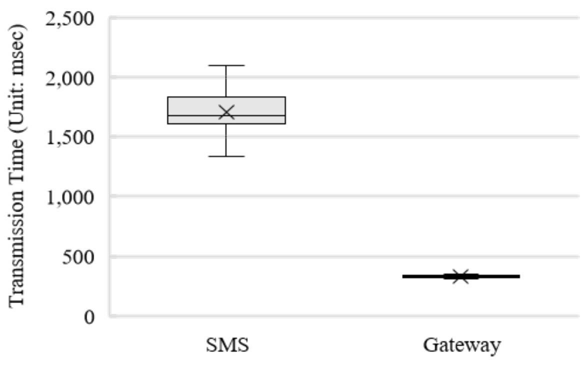


Fig. 8 Message transmission performance of SMS/Gateway.

The SMS in Fig. 2 introduces the Netty framework, an event-driven network framework, which offers an asynchronous multi-threaded design called EventLoop for the primary functions of message reception and transmission in the SMS. The developers of the SMS used the Channel structure provided by the Netty framework, but did not utilize the event handlers and message transmission channels offered by Netty. They implemented these as Handlers, Channel Queues, Global Queues, and Transmission Threads instead of using the built-in constructs provided by Netty, causing multiple threads to compete for access to the Global Queue and degrading overall performance in turn. Moreover, the use of synchronous sockets within the Transmission Threads further deteriorated performance. The root of all these issues was the failure to properly leverage the Netty framework. We can only guess that the reason for this was that the developers did not fully understand the proper usage of the framework and that requirement MC0 was not explicitly provided to them.

Fig. 9 illustrates the improved architecture using Netty's

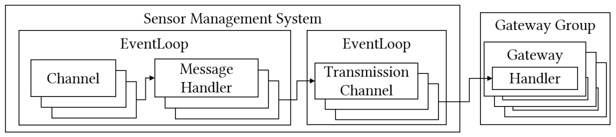


Fig. 9 Improved architecture using Netty's EventLoop design.

EventLoop design. The EventLoop handles all the I/O operations for a channel. EventLoops are created at a number twice the number of CPU cores, and the channels are uniformly allocated across these EventLoops. From the perspective of an individual channel, a single EventLoop is dedicated to it, thereby ensuring that the processing sequence of events occurring in each channel is maintained. When an event occurs in a channel, the responsible EventLoop forwards this event to the corresponding Message Handler. The Message Handler merges the data into a single message form and then relays that message through its assigned Transmission Channel. The Transmission Channel provided by Netty ultimately sends the message to the Gateway Handler. Consequently, an event occurring in a single channel is handled by a single EventLoop, Message Handler, and Transmission Channel, ensuring that message sequence integrity is maintained even in a multi-threading environment.

We used performance testing to verify whether the SMS with these architectural improvements meets requirement goal MC0, which was violated by the original SMS. Fig. 10 illustrates the result of the performance testing, i.e., message transmission performance, using a box-and-whisker plot. The x-axis indicates the type of architecture, and the y-axis displays the time needed to transmit 5,000 messages in milliseconds. The median performance for each architecture was 1,676 milliseconds the existing architecture and 316 milliseconds for the improved architecture, so performance was improved about

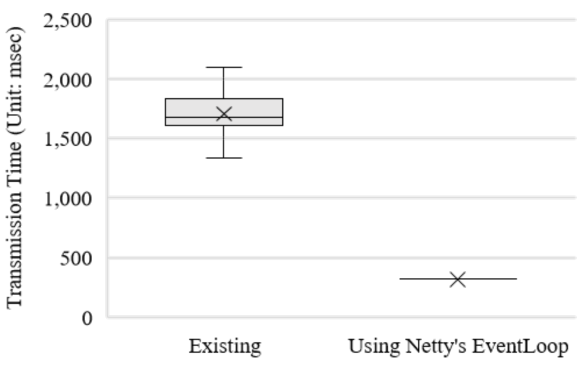


Fig. 10 Message transmission performance between the existing and the improved architecture of the SMS.

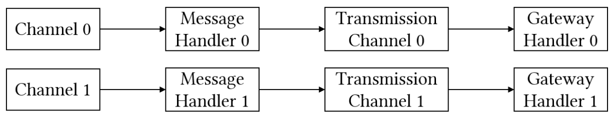


Fig. 11 Improved architecture model for model checking.

5.3 times. The interquartile ranges for each architecture were 222.2 and 1.8 for the existing architecture and the improved architecture, respectively, with 123.4 times of improvement. Both results show that 5,000 messages can be transmitted in one second, and therefore, we can safely conclude that the SMS with the improved architecture meets MC0.

We also modeled the SMS on the improved architecture in Promela and evaluated the model using the model checker SPIN to assure that the suggested architecture is free of message sequence inversion. The model of the SMS using the improved architecture is depicted in Fig. 11. By model-checking this architecture, we confirmed that no counterexamples violating the property of message sequence inversion were found. This is an expected result as messages for each channel are forwarded through their respective private lanes. Therefore, we conclude that the SMS with its architectural improvement also meets the goal TST0 (which is a super-goal of MC0).

V. Related Work

Requirements discovery in software engineering is the process of finding and understanding the needs and expectations of stakeholders in order to define the functional and non-functional requirements for a complicated system [13], [14]. This process is characterized by the systematic identification, analysis, and documentation of requirements and has evolved significantly over the years. Initially, it relied on traditional methods in which requirements were typically prioritized once using techniques such as interviews and manual document reviews [15]. Contemporary research in requirements discovery emphasizes non-functional requirements [16], [17] as well, shifting towards agile [18], [19], which involves iterative prioritization of requirements in each development cycle. It further explores the integration of automated tools and techniques, leveraging artificial intelligence and machine learning to streamline the discovery process [20], [21].

Studies aimed at effectively discovering the requirements of legacy systems utilize various viewpoints [22], [23]. Lee et. al. [24] proposed the PVRD methodology to create a proxy viewpoints model and provides a new way of discovering missing requirements while improving the requirements representation space through a new indexing structure that supports multiple viewpoints from many stakeholders in a large-scale complex software system. Elahi et. al. [25] introduced a vulnerability-centric requirements engineering framework that leverages goal modeling to analyze security.

To improve the quality of requirements discovered in the early phase, we iteratively conduct verification and requirements discovery using software testing and model checking whenever uncertain requirements are identified. Since the primary role of the EWS investigated in this study is to accurately detect earthquakes within given time constraints, and the delivery of earthquake alerts to people is integrated with a separate system, we focused on analyzing the requirements with an emphasis on the system's correctness and performance, which are of interest to the development team.

We needed an effective tool to analyze and manage the discovered goals. Goal-oriented requirements engineering (GORE) is a successful approach for addressing the challenges in goal achievement [26], [27]. GORE has been adapted and applied to many sub-topics within requirements engineering [28]. This approach emphasizes the use of goals to structure, specify, analyze, negotiate, document, and modify requirements [29]. There are various research efforts regarding goal modeling, specification, and reasoning in this area [30]–[33]. The methodologies for GORE basically involve gathering information from stakeholders, classifying ideas into functional and non-functional categories, and creating goal graphs to depict the dependencies among these ideas [34], [35].

The top-down approach aids in source code-based reverse engineering of large-scale distributed systems [36]. Cosma [37] presented an overview of distribution-related architecture using a dependency graph of the distributable features. Saives et al. [38] proposed automated partitioning for distributed behavioral identification using observations of the inputs and outputs of each component. For better efficiency of the EWS source code analysis, we carried out the code analysis in the following sequence: physical distribution relationships, network input/output, task behavior, and data dependencies, referencing existing research findings.

Lutz et al. [39] demonstrated the experience of discovering new requirements during the testing of mission-critical systems. These requirements newly discovered through the analysis of testing results usually involved complicated interface issues between software components. Several of the requirements involved fault protection, which is of special concern in mission-critical systems. We conducted testing and model checking to verify the goal model of the EWS. Following the findings of Lutz et al., we added the objective of finding new requirements in the goal model verification phase.

Due to the size of distributed systems, setting up a testing environment is challenging. A good way of addressing this problem is to make all components of a system configurable [40]. Gupta et al. [41] multiplexed all of the nodes in a distributed service configuration as virtual machines across a much smaller number of physical machines in a test harness. To replicate the actual deployment of the EWS with a small number of hosts, we parameterized the configurations of each component of the EWS, determining the parameter values when building each component as a virtual machine image.

While software testing is a cost-effective technique for the validation and verification of distributed systems, it is not suitable for confirming the correctness of the system, as it is typically based on observation and relies on the quality of the test cases, which are defined manually. On the other hand, model checking is able to perform rigorous and comprehensive verification, and thus, is necessarily required for verifying mission-critical or safety-critical systems [42]–[47]. Model checking can be applied to any software artifacts, provided that they are formally represented. Our approach utilized model checking to verify goal models and to verify the EWS system against the goals in question.

A framework-oriented approach provides better productivity and quality in software development, regardless of whether object-oriented, aspect-oriented, or service-oriented programming is used [48], [49]. It has been well adopted in system development, e.g., for medical systems [50] and multimedia applications [51]. Our case study is in line with the studies of Chang et al. [52] and Harrison et al. [53], who re-engineered legacy distributed systems using an open-source web framework and studied pattern-driven architectural partitioning to meet both functional requirements and non-functional quality attributes.

VI. Discussion and Contribution

We have presented a case study to identify non-functional requirements and a goal model of a mission-critical distributed system in operation by utilizing available software artifacts and V&V techniques. The EWS is a typical example of a distributed system in practice; Its components were developed by three or more organizations without applying software engineering principles (no requirements specification, no design documents, no formal quality assurance processes, etc.). Therefore, it has been extremely difficult to identify the sources of problems for a long time, even though there have been critical correctness/performance issues.

Through this case study:

1) We showed that it is possible to construct a meaningful goal-based requirements model from available software artifacts, no matter how stringent they are, by utilizing not only traditional requirements discovery techniques, such as interviews and document analysis, but also techniques from other disciplines, such as source code analysis, testing, and model checking.
2) We used interviews and document analysis to identify an initial goal model, which was incomplete and uncertain due to the lack of sources, and then refined the goal model through verification using software testing and model checking.
3) We were able to identify sub-goals in the goal model (and the source of violations) that were violated by the EWS using system testing.
4) We suggested new architectural solutions to satisfy the violated sub-goals in the goal model and confirmed that the solutions meet the sub-goals by using model checking.
5) The constructed goal model can be a valuable asset for the maintenance and evolution of the EWS system.

In this study, we learned that applying diverse existing software engineering techniques is very effective for discovering and managing requirements of mission-critical, distributed AI-enabled systems that were developed without applying rigorous software engineering principles. The five requirements discovery methods we utilized - interviews, document reviews, source code analysis, testing, and model checking - provided us with a high-quality goal model. Based on this goal model, we were able to derive systematic methods for system verification, maintenance, and evolution. Ultimately, these provided the effect of having been developed with rigorous application of requirements engineering from the beginning, even for the systems that were not initially developed using requirements engineering.

In the process of verifying the system w.r.t. the goal model and identifying the source of problems, we also learned the importance of accurate understanding and utilization of frameworks introduced for distributed systems development. The development of distributed systems often involves extensive adoption of third-party frameworks, which can help to achieve a sophisticated design and excellent performance. However, misunderstanding and misusing the framework (which is even worse when the requirements are not available) rather results in a deteriorating structure and performance of the system, which happens more often than not in practice. The goal model and the systematic testing approach we took helped us to identify such problems.

We believe this study could serve as an appealing case that demonstrates the importance of the disciplines of software engineering and requirements engineering to aid developers in the field of mission-critical distributed systems.

The following work remains as future work.

First, based on what we have learned from the experimental case study presented in this paper, it is necessary to develop a methodological approach that combines various existing requirements discovery & modeling methods, such as goal, scenario, viewpoints, and problem domain ontology model, (e.g., Onto-ActRE [54]), to enable requirements discovery and management in the operation of real-world business applications. Second, traditional software/requirements engineering approaches are not designed for data-centric AI-enabled systems. Therefore, it is necessary to design a requirements specification method for such systems. Machine learning models could learn the intent of users, the expected system behavior, and other environmental constraints from available datasets. This is part of the ongoing research efforts on AI engineering, and we plan to actively adopt this in order to deal with the aforementioned problems.

Acknowledgments

We thank Young-Woo Kwon and Jangsoo Lee at KNU for helping us understand the EWS. This work would not have been possible without their insights as developers.

The work of Yunja Choi and Youngsul Shin was supported by a National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (NRF-2021R1A5A1021944). Seok-Won Lee's work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) under the Artificial Intelligence Convergence Innovation Human Resources Development (IITP-2024-RS-2023-00255968) grant funded by the Korean government(MSIT).

References

[1] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of computer programming*, vol. 20, no. 1, pp. 3–50, 1993.

[2] A. Mavin, P. Wilkinson, S. Teufl, H. Femmer, J. Eckhardt, and J. Mund, "Does goal-oriented requirements engineering achieve its goal?" in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 174–183.

[3] D. Dell'Anna, F. Dalpiaz, and M. Dastani, "Validating goal models via bayesian networks." IEEE, 2018, pp. 39–46.

[4] Q. Kong, R. M. Allen, L. Schreier, and Y. W. Kwon, "Myshake: A smartphone seismic network for earthquake early warning and beyond," *Science advances*, vol. 2, no. 2, pp. e1 501 055–e1 501 055, 2016.

[5] S. Kumar, R. Vig, and P. Kapur, "Development of earthquake event detection technique based on STA/LTA algorithm for seismic alert system," *Journal of the Geological Society of India*, vol. 92, no. 6, pp. 679–686, 2018.

[6] Y. Fujinawa, Y. Rokugo, Y. Noda, Y. Mizui, M. Kobayashi, and E. Mizutani, "Development of application systems for earthquake early warning," *Journal of disaster research*, vol. 4, no. 4, pp. 546–556, 2009.

[7] H. Nakagawa, H. Shimada, and T. Tsuchiya, "Interactive goal model construction based on a flow of questions," *IEICE Transactions on Information and Systems*, vol. E103.D, no. 6, pp. 1309–1318, 2020.

[8] G. J. Holzmann, *The spin model checker: primer and reference manual*. Boston, MA: Addison-Wesley, 2003.

[9] Promela manual pages. [Online]. Available: https://spinroot.com/spin/Man/promela.html

[10] M. Ben-Ari, *Principles of the spin model checker*. London, UK: Springer Science & Business Media, 2008.

[11] B. Nuseibeh, "Weaving together requirements and architectures," *Computer (Long Beach, Calif.)*, vol. 34, no. 3, pp. 115–119, 2001.

[12] The Netty project. [Online]. Available: https://netty.io/

[13] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*. New York: John Wiley and Sons, 1998.

[14] I. Singh and S.-W. Lee, "RE_BBC: Requirements engineering in a blockchain-based cloud system: Its role in service-level agreement specification," *IEEE Software*, vol. 37, no. 5, pp. 7–12, 2020.

[15] L. Cao and B. Ramesh, "Agile requirements engineering practices: An empirical study," *IEEE software*, vol. 25, no. 1, pp. 60–67, 2008.

[16] S. Kopczyńska and J. Shi, "Can videos be used to communicate non-functional requirements? an early empirical investigation," in *2023 IEEE 31st International Requirements Engineering Conference (RE)*, 2023, pp. 305–310.

[17] F.-L. Li, J. Horkoff, J. Mylopoulos, R. S. S. Guizzardi, G. Guizzardi, A. Borgida, and L. Liu, "Non-functional requirements as qualities, with a spice of ontology," in *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, 2014, pp. 293–302.

[18] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in human behavior*, vol. 51, no. Part B, pp. 915–929, 2015.

[19] C. Palomares, X. Franch, C. Quer, P. Chatzipetrou, L. López, and T. Gorschek, "The state-of-practice in requirements elicitation: an extended interview study at 12 companies," *Requirements engineering*, vol. 26, no. 2, pp. 273–299, 2021.

[20] J. Deng, Z. Li, X. Zhou, and H. Xiao, "Nfrnet-lt:improving accuracy in extracting long-tailed non-functional requirements," in *2023 IEEE 31st International Requirements Engineering Conference (RE)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2023, pp. 355–356. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/RE57278.2023.00049

[21] M. Li, L. Shi, Y. Yang, and Q. Wang, "A deep multitask learning approach for requirements discovery and annotation from open forum." ACM, 2020, pp. 336–348.

[22] R. B. Cardoso, P. G. Brust-Renck, F. S. Fogliatto, G. L. Tortorella, and D. Samson, "User-centered requirement elicitation for the procurement of medical equipment used by different services and types of end-users," *Human factors and ergonomics in manufacturing & service industries*, vol. 32, no. 2, pp. 214–227, 2022.

[23] E. Kavakli, R. Sakellariou, I. Eleftheriou, and J. E. Mascolo, "Towards a multi-perspective methodology for big data requirements," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 5719–5720.

[24] S. W. Lee and D. C. Rine, "Missing requirements and relationship discovery through proxy viewpoints model," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, ser. SAC '04, 2004, p. 1513–1518. [Online]. Available: https://doi.org/10.1145/967900.968203

[25] G. Elahi, E. Yu, and N. Zannone, "A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities," *Requirements engineering*, vol. 15, no. 1, pp. 41–62, 2010.

[26] J. Horkoff and E. Yu, "Interactive goal model analysis for early requirements engineering," *Requirements engineering*, vol. 21, no. 1, pp. 29–61, 2016.

[27] S. Woldeamlak, A. Diabat, and D. Svetinovic, "Goal-oriented requirements engineering for research-intensive complex systems: A case study: Goal-oriented requirements engineering for rics," *Systems engineering*, vol. 19, no. 4, pp. 322–333, 2016.

[28] J. Horkoff, F. B. Aydemir, E. Cardoso, T. Li, A. Maté, E. Paja, M. Salnitri, L. Piras, J. Mylopoulos, and P. Giorgini, "Goal-oriented requirements engineering: an extended systematic mapping study," *Requirements engineering*, vol. 24, no. 2, pp. 133–160, 2019.

[29] A. van Lamsweerde, "Goal-oriented requirements engineering: a guided tour," in *Proceedings Fifth IEEE International Symposium on Requirements Engineering*, 2001, pp. 249–262.

[30] S. Anwer and N. Ikram, "Goal oriented requirement engineering: A critical study of techniques," in *2006 13th Asia Pacific Software Engineering Conference (APSEC'06)*, 2006, pp. 121–130.

[31] C. W. Mohammad, M. Shahid, and S. Z. Hussain, "Fuzzy attributed goal oriented software requirements analysis with multiple stakeholders," *International journal of information technology (Singapore. Online)*, vol. 13, no. 6, pp. 1–9, 2021.

[32] J. Hassine, D. Kroumi, and D. Amyot, "A game-theoretic approach to analyze interacting actors in grl goal models," *Requirements engineering*, vol. 26, no. 3, pp. 399–422, 2021.

[33] Q. Zhou, T. Li, and Y. Wang, "Assisting in requirements goal modeling: a hybrid approach based on machine learning and logical reasoning," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '22, 2022, p. 199–209. [Online]. Available: https://doi.org/10.1145/3550355.3552415

[34] A. Sainani, P. R. Anish, V. Joshi, and S. Ghaisas, "Extracting and classifying requirements from software engineering contracts," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*, 2020, pp. 147–157.

[35] E. Kavakli, "Goal-oriented requirements engineering: A unifying framework," *Requirements engineering*, vol. 6, no. 4, pp. 237–251, 2002.

[36] T. Katsimpa, Y. Panagis, E. Sakkopoulos, G. Tzimas, and A. Tsakalidis, "Application modeling using reverse engineering techniques," in *Proceedings of the 2006 ACM Symposium on Applied Computing*, 2006, p. 1250–1255. [Online]. Available: https://doi.org/10.1145/1141277.1141570

[37] D. C. Cosma, “Reverse engineering object-oriented distributed systems,” in *2010 IEEE International Conference on Software Maintenance*, 2010, pp. 1–6.

[38] J. Saives, G. Faraut, and J.-J. Lesage, “Automated partitioning of concurrent discrete-event systems for distributed behavioral identification,” *IEEE transactions on automation science and engineering*, vol. 15, no. 2, pp. 832–841, 2018.

[39] R. Lutz and I. Mikulski, “Requirements discovery during the testing of safety-critical software,” in *25th International Conference on Software Engineering, 2003. Proceedings.*, 2003, pp. 578–583.

[40] P. Maddox, “Testing a distributed system: Testing a distributed system can be trying even under the best of circumstances.” *Queue*, vol. 13, no. 7, p. 10–15, jul 2015. [Online]. Available: https://doi.org/10.1145/2800695.2800697

[41] D. Gupta, K. V. Vishwanath, M. McNett, A. Vahdat, K. Yocum, A. Snoeren, and G. M. Voelker, “Diecast: Testing distributed systems with an accurate scale model,” *ACM Trans. Comput. Syst.*, vol. 29, no. 2, may 2011. [Online]. Available: https://doi.org/10.1145/1963559.1963560

[42] W. B. Daszczuk, *Integrated Model of Distributed Systems*, 1st ed. Cham: Springer International Publishing, 2020;2019;, vol. 817.

[43] E.-Y. Kang, L. Huang, and D. Mu, “Formal verification of energy and timed requirements for a cooperative automotive system,” in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC ’18, 2018, p. 1492–1499. [Online]. Available: https://doi.org/10.1145/3167132.3167291

[44] M. Camilli, “Petri nets state space analysis in the cloud,” in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE ’12. IEEE Press, 2012, p. 1638–1640.

[45] L. Di Stefano, R. De Nicola, and O. Inverso, “Verification of distributed systems via sequential emulation,” *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 3, mar 2022. [Online]. Available: https://doi.org/10.1145/3490387

[46] O. Adesina, T. C. Lethbridge, and S. Somé, “Optimizing hierarchical, concurrent state machines in umple for model checking,” in *Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS ’19. IEEE Press, 2021, p. 524–532. [Online]. Available: https://doi.org/10.1109/MODELS-C.2019.00082

[47] D. Wang, W. Dou, Y. Gao, C. Wu, J. Wei, and T. Huang, “Model checking guided testing for distributed systems,” in *Proceedings of the Eighteenth European Conference on Computer Systems*, ser. EuroSys ’23, 2023, p. 127–143. [Online]. Available: https://doi.org/10.1145/3552326.3587442

[48] M. M. Arimoto, M. I. Cagnin, and V. V. de Camargo, “Version control in crosscutting framework-based development,” in *Proceedings of the 2008 ACM Symposium on Applied Computing*, ser. SAC ’08, 2008, p. 753–758. [Online]. Available: https://doi.org/10.1145/1363686.1363862

[49] C. Chen, A. Zaidman, and H.-G. Gross, “A framework-based runtime monitoring approach for service-oriented software systems,” in *Proceedings of the International Workshop on Quality Assurance for Service-Based Applications*, ser. QASBA ’11, 2011, p. 17–20. [Online]. Available: https://doi.org/10.1145/2031746.2031752

[50] P. Bellagente, C. Crema, A. Depari, A. Flammini, G. Lenzi, and S. Rinaldi, “Framework-oriented approach to ease the development of ambient assisted-living systems,” *IEEE Systems Journal*, vol. 13, no. 4, pp. 4421–4432, 2019.

[51] J. Lin, J. Drake, H. Kim, and E. Song, “A framework-based approach for interactive multimedia application development,” in *Proceedings of the 2012 ACM Research in Applied Computation Symposium*, ser. RACS ’12, 2012, p. 364–370. [Online]. Available: https://doi.org/10.1145/2401603.2401683

[52] C.-H. Chang, C.-W. Lu, W. C. Chu, N.-L. Hsueh, and C.-S. Koong, “A case study of pattern-based software framework to improve the quality of software development,” in *Proceedings of the 2009 ACM Symposium on Applied Computing*, ser. SAC ’09, 2009, p. 443–447. [Online]. Available: https://doi.org/10.1145/1529282.1529379

[53] N. Harrison and P. Avgeriou, “Pattern-driven architectural partitioning: Balancing functional and non-functional requirements,” in *2007 Second International Conference on Digital Telecommunications (ICDT’07)*, 2007, pp. 21–21.

[54] S. Lee and R. A. Gandhi, “Ontology-based active requirements engineering framework,” in *In Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC ”05)*, 2005.