IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 15, 2022, accepted February 28, 2022, date of publication March 22, 2022, date of current version March 29, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3161545

Goal-Oriented Software Design Reviews

MICHIYO WAKIMOTO AND SHUJI MORISAKI, (Member, IEEE)
Graduate School of Informatics, Nagoya University, Nagoya, Aichi 464-8601, Japan
Corresponding author: Michiyo Wakimoto (wakimoto.michiyo@g.mbox.nagoya-u.ac.jp)

ABSTRACT Some software requirements are omitted or ambiguous depending on the design context, although these requirements would not necessarily be omitted or ambiguous when viewed as requirements alone. The design context sometimes causes inconsistencies among implementations that realize the same requirement. Existing detection and analysis methods do not limit evaluation of review materials to implementations of context-dependent design. An evaluation technique that limits the evaluated parts to the parts describing context-dependent design implementations is expected to be efficient. This paper proposes a method for detecting inconsistent implementations (context-dependent requirement defects) caused by context-dependent requirement omissions and ambiguities in design reviews. The proposed method defines goal-oriented check items for design review using a goal tree obtained by goal-oriented requirements analysis. Reviewers use the goal-oriented check items to detect inconsistent implementations that realize the same requirement. This paper also evaluates the proposed method through a case study. The results of the case study showed that the proposed method defined five goal-oriented check items and that reviewers detected 24 context-dependent requirement defects with goal-oriented check items. The results also showed that the sum of the estimated additional effort to define goal-oriented check items and perform design reviews with goal-oriented check items was 19.6 person-hours. Furthermore, the results showed that an engineer with general skills and knowledge of software development but without system-specific skills and knowledge could define a goal tree and the corresponding goal-oriented check items.

INDEX TERMS Context-dependent requirement (CDR), goal-oriented reviews, software reviews, software quality.

I. INTRODUCTION

Inappropriate requirements can consume substantial rework effort in subsequent development activities. In particular, omissions and ambiguities in requirements may lead to extensive changes and corrections in the subsequent development activities. The causes of requirement omissions include missing functionality, missing performance, missing interface, and missing environment [1]. Ambiguity enables multiple interpretations of the requirements document [2]. Various approaches and methods to reduce omissions and ambiguities in requirements have been proposed. Software review is one such static analysis technique for the early detection of defects, including omissions and ambiguities, and does not require program execution [3], [4]. Software review is a visual software-artifact evaluation technique to detect anomalies, defects, errors, or deviations from specifications or standards [5]. Perspective-based reading [6] is a reading technique for software review from the perspectives of the stakeholders; it is designed to reduce omissions and ambiguities in requirements through multiple perspectives. The perspectives provide reviewers guides for finding defects from the viewpoint of stakeholders such as project managers, users, and testers. Goal-oriented requirements analysis [7] prevents missing requirements and facilitates requirements decomposition [8], [9]. Requirements decomposition increases the requirements coverage by defining goals at various levels of abstraction [8].

Some requirements are omitted or ambiguous depending on the design context, although these requirements would not necessarily be omitted or ambiguous when viewed as requirements alone. The design context is determined during the software design activity. This paper refers to such omissions or ambiguities in requirements caused by design context as context-dependent requirement issues (CDRIs). A CDRI occurs when two or more different implementations realize the same requirement because the requirements are defined before the implementations are defined. Although

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

each implementation is adequately supported by the requirement, the implementations are not always consistent. For example, the requirement ''The data are exchanged with files. The fields in the file must be separated by a line break'' can be realized by two implementations: one implementation for writing a data file and another implementation for reading the data file. If the two implementations are realized on the same operating system, no CDRI occurs because the line-break characters are the same between the implementations. However, a CDRI occurs if the two implementations are realized on different operating systems and do not consider the line-break characters for the other operating system. Specifically, if the implementation for writing a file is realized with UNIX (LF for a line-break character) and the implementation for reading the file is realized with Windows (CR and LF for a line-break character), the fields will not be separated properly despite each of the two implementations realize the requirement accurately. In this case, the design context is the line-break characters for the operating systems.

Feasibility or impact analysis, which can help reviewers find such design contexts during the requirement process, requires extensive effort because the analyses check all implementations: not only two or more implementations, which realize the same requirement, but also a single implementation. Identifying two or more implementations, which are supported by a single requirement and checking consistencies among them in the design review can help detect inconsistencies caused by CDRIs.

To the best of our knowledge, no specific approach or method to detect CDRIs or context-dependent requirement (CDR) defects caused by CDRIs has been proposed. This paper proposes a design review method to identify such inconsistencies among implementations realizing the same requirement by using a goal tree obtained by goal-oriented requirements analysis. The proposed method defines check items to find inconsistencies in the implementations, where the check items are created from the goal tree. This paper also evaluates the proposed method through a case study with two criteria. First, the evaluation investigates whether the check items for design review can be defined from the goal tree and then whether the check items can detect CDR defects. Second, the evaluation investigates whether the proposed method reduces the estimated rework effort to correct defects.

This paper is structured as follows. The related research and proposed method are described in Section II and Section III, respectively. Section IV describes a case study. Section V discusses the results, and Section VI summarizes this paper.

II. RELATED RESEARCH

Guided reviews are one approach to detecting defects caused by omissions or ambiguities in requirements. Guided reviews help reviewers comprehensively detect severe defects, including omissions or ambiguities, by providing detailed instructions, procedures, and hints. Many studies have reported on the effectiveness of guided reviews [1], [6], [10]–[17]. Typical techniques of guided reviews are checklist-based reading (CBR) [3], perspective-based reading (PBR) [12], defect-based reading (DBR) [11], usage-based reading (UBR) [13], and traceability-based reading [18]. CBR is a reading technique in which reviewers use a list of questions to help them understand what defects to examine [14]. PBR [6], [19], [20] is a scenario-based reading (SBR) [11] that defines the perspectives of the stakeholders and assigns the perspectives to reviewers. DBR is a SBR that focuses on detecting specific types of defects [11], [14]. UBR prioritizes the use cases and detects the most critical defects in the target materials along with the prioritized use cases [14], [18]. However, these reading techniques do not require that guides including checklists and scenarios verify inconsistencies among different implementations for the same requirement.

Goal-oriented requirements analysis [7], [21] is one of the methods to reduce omissions or ambiguities in requirements. Goal-oriented requirements analysis defines software requirements by clarifying the structured goals of the software. Goal-oriented requirements analysis also clarifies the background and necessities for requirements, facilitates requirements analysis discussions, and enhances the validation and tracking of changes to the requirements. Many goal-oriented requirements analysis methods have been studied, including the KAOS method [22]–[26], the i* framework [27]–[31], and the NFR framework [32]–[35]. However, detecting omissions or ambiguities in requirements caused by the design context determined in the design process is difficult because goal-oriented requirements analysis is performed during the requirements processes.

Traceability between requirements and design elements can verify that the requirements have been implemented as design elements in the design document [36]–[41]. Traceability studies have strongly focused on requirements traceability, with the objective of studying how to describe and follow requirements in both the forward and backward directions [37], [40]. Traceability is an effective guide to detect CDRIs; however, it is unclear whether traceability can examine consistencies among implementations. Although two methods [42], [43] check consistencies between requirements and design documents, both of them check consistencies between different types of UML documents. Thus, they cannot always detect inconsistencies among the implementations in the same UML document.

Change impact analysis identifies where the changes affect [44]–[46], estimates the effort for implementing a change request [47], [48], and predicts necessary regression tests according to the set of changes [49]. However, change impact analysis cannot always detect inconsistencies among implementations. Automotive-SPICE [50] recommends analyzing the operational (execution) environment, including platforms, to analyze the feasibility of the requirements. Such operational environment analysis can detect implementation inconsistencies. However, it does not explicitly refer to detecting inconsistencies among implementations.

III. PROPOSED METHOD

A. PREREQUISITE

In the proposed method, reviewers attempt to detect inconsistencies among design implementations and ensure that the implementations satisfy the goal using goal-oriented check items. This paper refers to the inconsistencies as CDR defects. CDR defects are caused by CDRIs. The proposed method defines goal-oriented check items taking a goal tree as input. The goal tree provides traceability links from high-level strategic objectives to low-level technical requirements [8]. The proposed method adds goal-oriented check items to the leaf nodes of the goal tree, which is created by goal-oriented requirements analysis. The goal tree consists of the top goal and subgoals.The top goal is the root node of the goal tree. The root node has a label describing the objective or state that the system should achieve. The top goal is decomposed into one or more subgoals (child nodes) because, without the decomposition, a goal tree may not provide technical requirements. The goal-oriented check items defined from the subgoals that do not satisfy the desired requirement cannot detect inconsistencies among context-dependent implementations. Thus, the decomposition should be performed carefully. The goal-oriented requirements analysis [24], [33] categorizes goals into three categories: functional requirements, non-functional requirements, and external constraints. Thus, the top goal and subgoals for the proposed method can be categorized into three categories. The node has a label describing the purpose or the status required to achieve the parent goal (parent node). Subgoals are recursively decomposed into sub-subgoals. The label description is a prescriptive statement of intent that the system should satisfy [22].

B. PROCEDURE

1) Identify the goal tree (top goal and subgoals). If goal-oriented requirements analysis has created a goal tree in advance (e.g., requirements analysis), the goal tree is reused. If the goal tree does not exist, the analyst describes the goal of the system as the label of the top goal G. The analyst then decomposes the top goal G into subgoals G1, G2,..., G*m*. The analyst creates a simple label for each subgoal and adds the subgoals as child nodes of the top goal. The analyst decomposes the subgoals (G1, G2, ..., G*m*) into sub-subgoals (G1.1, G1.2, G1.3, ..., G2.1, G2.2, ..., G*m*.1, G *m*.2, ...) and continues decomposing the subgoals until the subgoals are complete, consistent, and minimal.
2) Prune unnecessary subgoal nodes. The analyst selects and prunes unnecessary subgoal nodes, which do not need to be broken down further for consideration, such as duplicated subgoals. After pruning, each leaf node of the goal tree is marked as a leaf subgoal node.
3) Define goal-oriented check items. Goal-oriented check items verify whether the design implementations are consistent and satisfy the corresponding subgoal. Goal-oriented check items are determined by two or more implementations that realize the same subgoal. The analyst defines one or more goal-oriented check items for each of the leaf subgoal nodes except pruned subgoals. The analyst then adds goal-oriented check items as child nodes of the leaf subgoal nodes. Fig. 1 shows an example goal tree. Goal nodes and goal-oriented check-item nodes are labeled with symbols G and C, respectively. The pruned subgoals are indicated by red circles, as shown in G1.1.2 and G2.
4) Perform goal-oriented software design review. Reviewers use the goal-oriented check items to attempt to detect inconsistencies among implementations realizing the same requirement.

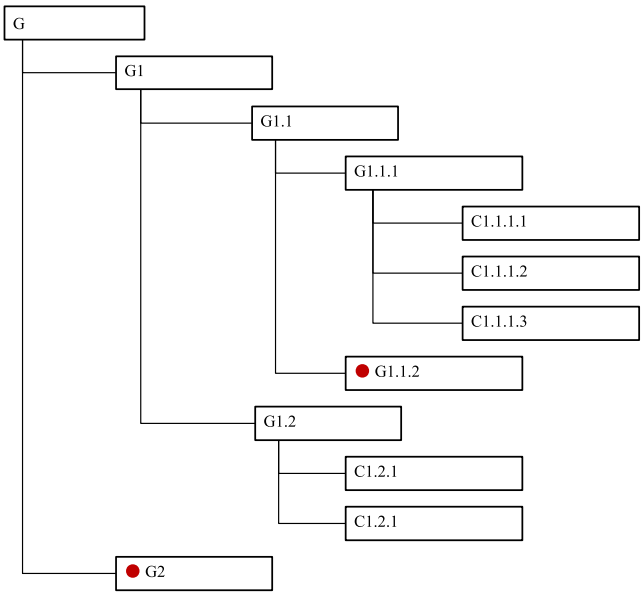

FIGURE 1. An example of a goal tree. Red circles represent leaf subgoal nodes.

C. EXAMPLE OF A GOAL TREE AND CHECK ITEMS

1) OVERVIEW OF AN EXAMPLE SYSTEM

This subsection presents a goal tree and the corresponding goal-oriented check items for an example system. Table 1 and Fig. 2 show an overview of the example system. The design implementations for the same subgoal can differ among the three units because the units had different developers.

TABLE 1. Overview of an example system.

Name	Electric kettle
Goal	The electric kettle heats water and keeps the water temperature constant.
Architecture and developers	The system consists of a sensor unit, control unit, and heating unit. Each unit was developed by different developers (no developer developed two or more units).

2) PROCEDURE

Fig. 3 shows the goal tree and goal-oriented check items. The procedure is detailed as follows.

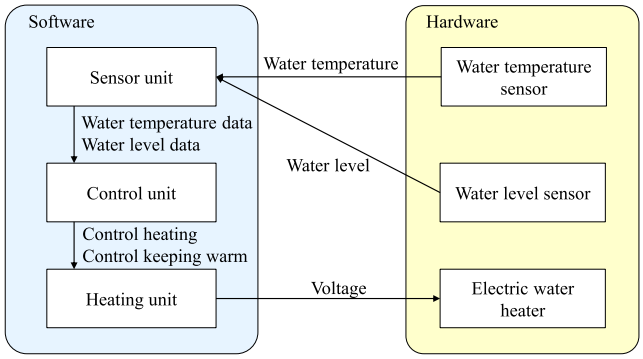

FIGURE 2. Architecture of an example system.

Step (1). The analyst identifies the goal tree. The identified goal tree consists of the following:

Top goal G: The electric kettle heats water and keeps the water temperature constant.

The analyst decomposes the top goal G into the following subgoals:

G1: The electric kettle can heat the water if the water temperature is less than the threshold.

G2: The electric kettle can keep the water temperature constant if the water temperature is greater than or equal to the threshold.

G3: The electric kettle cannot heat the water or keep the water temperature constant if the water level is less than the minimum level.

The analyst decomposes subgoal G1 into the following subgoals:

G1.1: The electric kettle can heat the water with electricity.

G1.2: The electric kettle can measure the water temperature periodically.

The analyst decomposes subgoal G2 into the following subgoals:

G2.1: The electric kettle can keep the water temperature constant using electricity.

G2.2: The electric kettle can measure the water temperature periodically.

The analyst decomposes subgoal G3 into the following subgoals:

G3.1: The electric kettle can measure the water level periodically.

G3.2: The electric kettle cannot heat the water or keep the water temperature constant if the water level is less than the minimum level.

Step (2). The analyst prunes unnecessary subgoal node G2.2 because G2.2 duplicates G1.2.

Step (3). The analyst defines goal-oriented check items as child nodes of the subgoals. The analyst defines the following goal-oriented check items from G1.1:

C1.1.1: Is the voltage control method during heating specified? If one or more descriptions are specified, are they consistent?

The analyst defines the following goal-oriented check items from G1.2:

C1.2.1: Is the water temperature measurement period specified? If one or more descriptions are specified, are they consistent?

C1.2.2: Is the unit system of the water temperature specified? If one or more descriptions are specified, are they consistent?

The analyst defines the following goal-oriented check item from G2.1:

C2.1.1: Is the voltage control for keeping the water temperature constant specified? If one or more descriptions are specified, are they consistent?

The analyst defines the following goal-oriented check items from G3.1:

C3.1.1: Is the water level measurement period specified? If one or more descriptions are specified, are they consistent?

C3.1.2: Is the unit system of the water level specified? If one or more descriptions are specified, are they consistent?

The analyst defines the following goal-oriented check item from G3.2:

C3.2.1: Is the procedure to stop heating or to stop keeping the water temperature constant specified? If one or more descriptions are specified, are they consistent?

Step (4). The reviewer performs goal-oriented software design review using the goal-oriented check items. For example, in G1.2, the developers of the sensor unit considered and defined the water temperature in Fahrenheit, whereas the developers of the control unit considered and defined the temperature in Celsius. The reviewer can detect this inconsistency between the definitions and implementation with goal-oriented check item C1.2.2.

IV. CASE STUDY

A. GOAL

The goal of the case study is to investigate the effectiveness and efficiency of the proposed method. The case study was conducted with a commercial software system. An overview of the commercial software system is described in Subsection IV.B. The case study evaluated whether the proposed method could define goal-oriented check items, whether the proposed method could detect CDR defects, and whether detecting CDR defects in design reviews contributed to a reduction of the rework effort for correcting the defects.

B. SYSTEM CONTEXT

We selected the subsystems of System A developed in a Japanese software development Company S for this case study. System A was a communication network control system. Table 2 and Fig. 4 shows the details. The development period was from April 2017 to March 2019. System A consisted of 12 subsystems. Each subsystem was developed from scratch. The number of developers for each subsystem varied

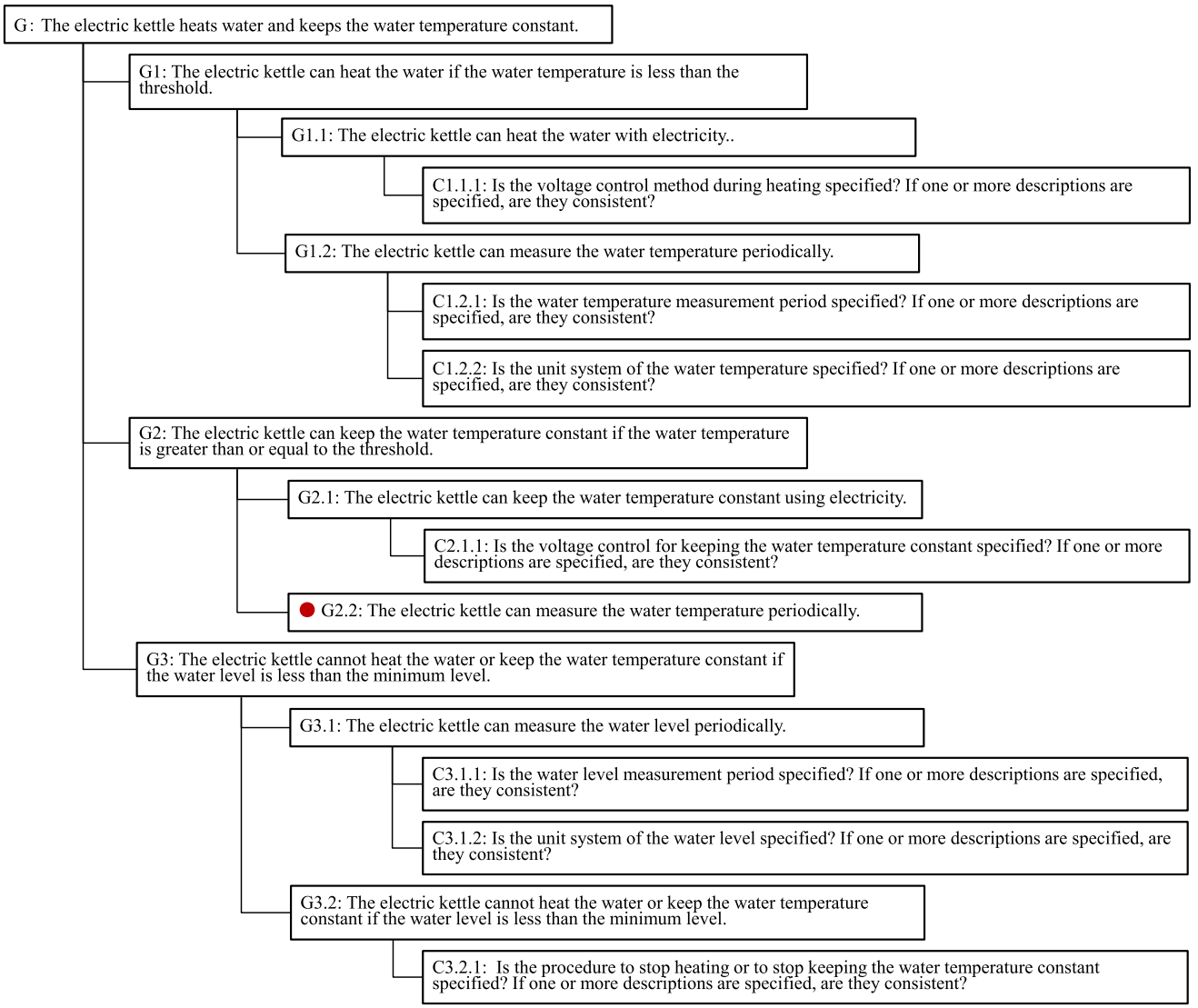

FIGURE 3. A goal tree for the example system.

TABLE 2. Overview of system A.

Name	Communication network control system
Goal	The system controls emergency communication for human safety. The system enables every terminal in the network to communicate with other terminals in the network.
Architecture	• System A consisted of monitoring control unit, communication control unit, and data transmission unit. • The monitoring control unit reused another software whose reliability was proved in another system in operation. No severe defects had not been found in the original software.

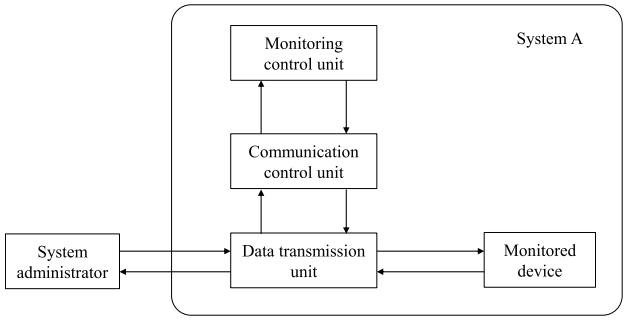

FIGURE 4. Architecture of system A.

from three to seven. The number of years of software development experience of the developers varied from 2 to 25 years. The lines of source code of the subsystems varied from 3100 to 8400 lines in C language.

The standard software development process was based on the waterfall model and followed the process areas Organizational Process Definition (OPD) and Integrated Project Management (IPM) defined in CMMI-DEV V.1.3 [51]. The

TABLE 3. Metrics for the evaluation.

	Name	Description
cH	Effort to define goal-oriented check items	Person-hours to identify a goal tree and define goal-oriented check items
grH	Effort for goal-oriented software design reviews	Person-hours to perform design reviews with the goal-oriented check items
gwH	Estimated additional rework effort	Difference between the sum of the estimated effort (person-hours) for investigating, fixing, and regression testing and the sum of the effort for fixing defects detected in the goal-oriented software design reviews
grD	Number of CDR defects detected in goal-oriented software design reviews	Number of CDR defects detected in design reviews with goal-oriented check items
srD	Number of CDR defects detected in standard design reviews	Number of CDR defects among defects detected in the standard design reviews
gtD	Number of CDR defects detected in subsequent testing	Number of CDR defects, which were overlooked in design reviews and detected in subsequent software testing

standard process also defined software measurements and metrics. For each software development, the standard process required that the defects detected in reviews should be recorded in a defect list and that the defects detected in testing should also be recorded. The standard software development process required that each project perform design reviews (standard design reviews), record the design review logs (meeting minutes), and use the standard design review checklists. The standard software development process of Company S also required that each reviewer complete the review training and have detailed knowledge of the system domain to participate in the review.

C. METRICS

The metrics cH, grH, gwH, grD, srD, and gtD (Table 3) were measured for this evaluation in addition to the metrics in the standard software development process of Company S. These metrics are defined in Table 3. The metrics cH and grH are efforts for the preparation of the proposed method; metrics grD, srD, and gtD are the number of detected defects. Note that the metric gwH was the estimated additional rework effort (person-hours) that would be needed if the CDR defects detected in the proposed method were overlooked in goal-oriented software design reviews and detected and corrected in subsequent software testing.

D. EVALUATION AND PROCEDURE

We selected four subsystems from the 12 subsystems of System A. We selected two subsystems 1a and 2a from the four subsystems for Evaluations 1 and 2. For Evaluation 3, from the remaining subsystems, we selected subsystem 1b with a goal similar to that of subsystem 1a. Similarly, we selected subsystem 2b with a goal similar to that of subsystem 2a.

1) EVALUATION 1: CAN AN ANALYST DEFINE GOAL-ORIENTED CHECK ITEMS USING THE PROPOSED METHOD?

Evaluation 1 evaluated whether an analyst (engineer) could identify a goal tree and define the corresponding goal-oriented check items. Following the steps in Subsection III.B, the analyst identified goal trees and defined goal-oriented check items for subsystems 1a and 2a. The analyst is a quality assurance engineer and one of the authors.

2) EVALUATION 2: CAN THE PROPOSED METHOD DETECT CDR DEFECTS AND REDUCE THE DEFECT CORRECTION EFFORT?

Evaluation 2 consisted of the following evaluations:

> Evaluation 2.1: Can reviewers detect CDR defects in goal-oriented software design reviews?
> Evaluation 2.2: Can the proposed method reduce the estimated additional rework effort to correct CDR defects?

Evaluation 2.1 measured the number of CDR defects detected in goal-oriented software design reviews (grD). In addition, Evaluation 2.1 measured the number of CDR defects detected in subsequent testing (gtD) because overlooked CDR defects in goal-oriented software design reviews could be detected in subsequent software testing. Evaluation 2.2 measured the effort to define goal-oriented check items (cH), the effort for goal-oriented software design reviews (grH), and the estimated additional rework effort (gwH) to investigate whether the proposed method required less effort than the standard design reviews and the subsequent testing defined by the standard process. Specifically, Evaluation 2.1 deemed that the proposed method was feasible if CDR defects were detected in goal-oriented software design reviews ($grD > 0$) and the number of CDR defects detected in subsequent testing (gtD) was sufficiently small. In Evaluation 2.2, if the sum of the effort to define goal-oriented check items and the effort for goal-oriented software design reviews was smaller than the estimated additional rework effort ($cH + grH < gwH$), the proposed method was efficient.

Reviewers performed goal-oriented software design reviews for subsystems 1a and 2a using the goal-oriented check items defined in Evaluation 1. The goal-oriented software design reviews were performed in addition to the standard design reviews. After the goal-oriented software design reviews, the subsequent development activities, including software testing, were performed according to the standard

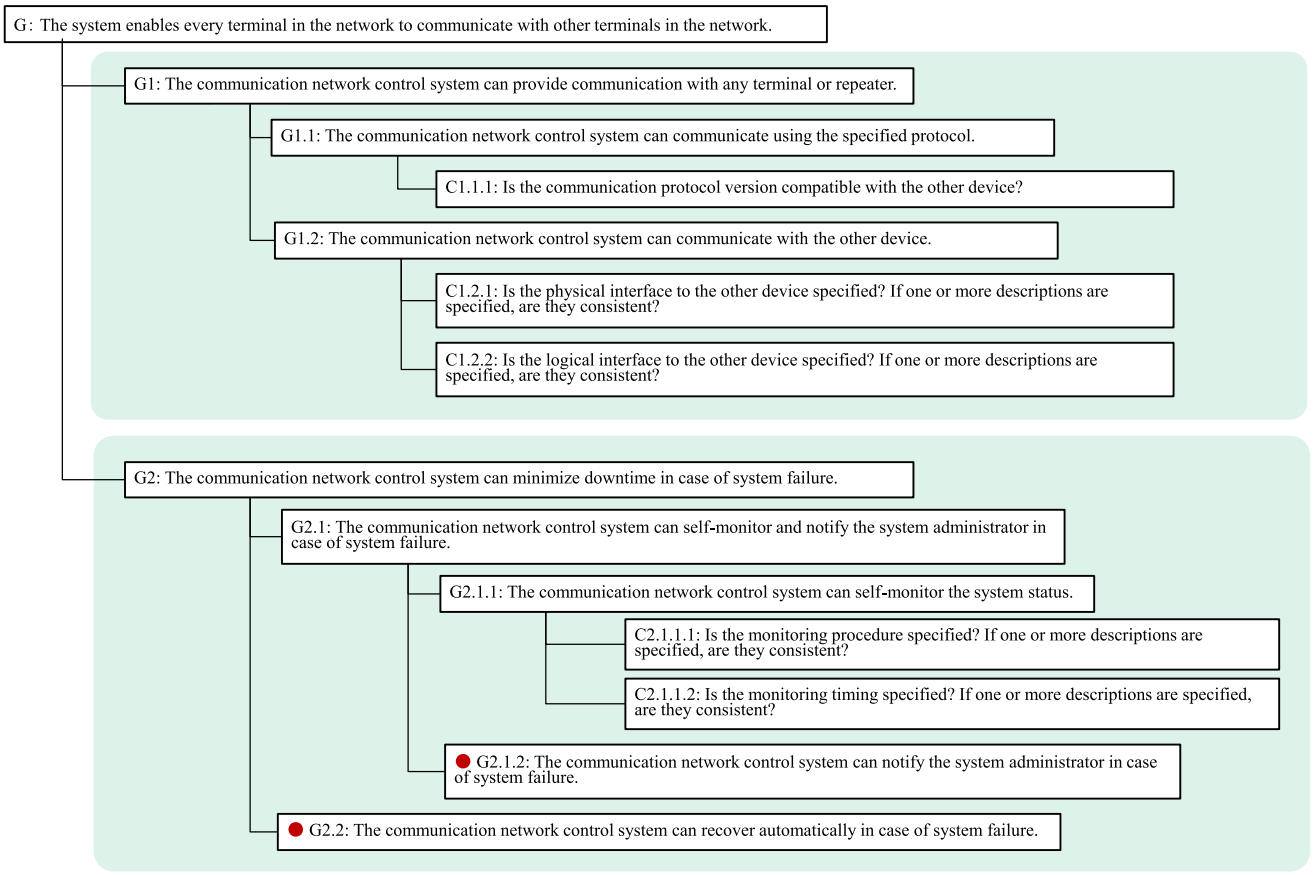

FIGURE 5. The goal tree and goal-oriented check items for system A.

software development process. The analyst measured and recorded the metrics in Table 3 (excluding srD). The analyst then categorized the CDR defects detected in goal-oriented software design reviews and the subsequent testing into defect groups corresponding to goal-oriented check items defined in Evaluation 1.

3) EVALUATION 3: ARE CDR DEFECTS DETECTED IN OTHER SIMILAR SUBSYSTEMS?

Evaluation 3 measured the number of CDR defects detected in the standard design reviews (srD) and subsequent testing (gtD) for subsystems 1b and 2b to investigate the applicability of the proposed method in other subsystems. Specifically, Evaluation 3 measured srD to investigate whether the standard design reviews detected CDR defects and gtD to investigate whether the standard design reviews overlooked CDR defects.

Evaluation 3 deemed that CDR defects existed in design documents for subsystems 1b and 2b if CDR defects were detected in the standard design reviews and/or subsequent testing (srD + gtD > 0). If CDR defects were present and the proposed method was carried out, goal-oriented software design reviews could possibly detect CDR defects. If CDR defects were detected in the standard design reviews and the number of detected CDR defects in the subsequent testing was sufficiently small (srD > 0 and srD >> gtD), the standard design reviews were considered to be able to detect most of CDR defects. If CDR defects were not detected in the standard design reviews or in subsequent testing (srD = gtD = 0), CDR defects were not considered to have been injected in the design documents.

Reviewers performed the standard design reviews for subsystems 1b and 2b. After the standard design reviews, the subsequent development activities, including software testing, were performed according to the standard software development process. The analyst categorized the CDR defects detected in the standard design reviews and subsequent testing into defect groups corresponding to the goal-oriented check items defined in Evaluation 1.

E. RESULTS

1) RESULTS OF EVALUATION 1

Fig. 5 shows the goal tree and the goal-oriented check items, which the analyst identified and defined. In Fig. 5, subgoals G1 and G2 were realized by subsystems 1a and 2a, respectively. Notably, subgoals G2.1.2 and G2.2 were not broken down further because these subgoals were realized by

the reused software whose reliability was already proven in another system in operation.

2) RESULTS OF EVALUATION 2

As shown in Table 4, the value of grD was six for subsystem 1a and eighteen for subsystem 2a. For subsystem 1a, the sum of cH and grH was 8.3 and the value of gwH was 42.0. The sum of cH and grH was 19.6% of the value of gwH. For subsystem 2a, the sum of cH and grH was 9.0 and the value of gwH was 54.0. The sum of cH and grH was 16.7% of the value of gwH.

TABLE 4. Results for evaluation 2.

Subsystem	cH	grH	gwH	grD	gtD
1a	1.0	7.3	42.0	6	0
2a	1.0	8.0	54.0	18	0

Table 5 shows the goal-oriented check items and the number of defects categorized as the defect groups, which could be detected with the goal-oriented check items for subsystem 1a. For subsystem 1a, the reviewer detected four defects for C1.1.1 and two defects for C1.2.2. No CDR defect was detected in subsequent software testing for subsystem 1a.

TABLE 5. Number of defects for goal-oriented check items for subsystem 1a.

	Goal-oriented check item	grD	gtD
C1.1.1	Is the communication protocol version compatible with the other device?	4	0
C1.2.1	Is the physical interface to the other device specified? If one or more descriptions are specified, are they consistent?	0	0
C1.2.2	Is the logical interface to the other device specified? If one or more descriptions are specified, are they consistent?	2	0
Total		6	0

Table 6 shows the goal-oriented check items and the number of defects in defect groups corresponding to the goal-oriented check items for subsystem 2a. For subsystem 2a, the reviewer detected thirteen defects for C2.1.1.1 and five defects for C2.1.1.2. No CDR defects were detected in subsequent software testing for subsystem 2a.

3) RESULTS OF EVALUATION 3

As shown in Table 7, the value of srD was two for subsystem 1b and five for subsystem 2b. Table 7 also shows that the value of gtD was one for subsystem 1b and two for subsystem 2b.

Table 8 shows the goal-oriented check items for subsystem 1b and the number of detected defects for the check items. For subsystem 1b, the reviewer detected two CDR defects for C1.2.1 in the standard design reviews. Subsequent testing detected one CDR defect for C1.1.1.

Table 9 shows the goal-oriented check items for subsystem 2b and the number of detected defects for the check items.

TABLE 6. Number of defects for goal-oriented check items for subsystem 2a.

	Goal-oriented check item	grD	gtD
C2.1.1.1	Is the monitoring method specified? If one or more descriptions are specified, are they consistent?	13	0
C2.1.1.2	Is the monitoring timing specified? If one or more descriptions are specified, are they consistent?	5	0
Total		18	0

TABLE 7. Results for evaluation 3.

Subsystem	srD	gtD
1b	2	1
2b	5	2

TABLE 8. Number of defects for goal-oriented check items for subsystem 1b.

	Goal-oriented check item	srD	gtD
C1.1.1	Is the communication protocol version compatible with the other device?	0	1
C1.2.1	Is the physical interface to the other device specified? If one or more descriptions are specified, are they consistent?	2	0
C1.2.2	Is the logical interface to the other device specified? If one or more descriptions are specified, are they consistent?	0	0
Total		2	1

TABLE 9. Number of defects for goal-oriented check items for subsystem 2b.

	Goal-oriented check item	srD	gtD
C2.1.1.1	Is the monitoring method specified? If one or more descriptions are specified, are they consistent?	2	1
C2.1.1.2	Is the monitoring timing specified? If one or more descriptions are specified, are they consistent?	3	1
Total		5	2

For subsystem 2b, the reviewer detected two CDR defects for C2.1.1.1 and three CDR defects for C2.1.1.2 in the standard design reviews. Subsequent software testing detected one CDR defect for C2.1.1.1 and one CDR defect for C2.1.1.2.

V. DISCUSSION

A. EVALUATION RESULTS

In Evaluation 1, a quality assurance engineer defined both a goal tree and the corresponding goal-oriented check items without additional explanations for System A. This indicated that the proposed method does not require a domain expert as an analyst. In discussion, another engineer of the case study said, "Although the quality assurance engineer is not a member of the development team, the engineer could define the goal tree and the corresponding goal-oriented check items.

For future development, I suppose that engineers with software quality assurance skills can define them.''

Evaluation 2.1 showed that the reviewers could perform goal-oriented software design reviews and detect CDR defects. In addition, Evaluation 2.2 showed that the estimated additional rework effort for the detected defects in subsequent testing was reduced by the defects detected in the goal-oriented software design reviews. The case study results showed that the CDR defects were detected by goal-oriented software design reviews in both subsystems 1a and 2a. In subsequent activities, including testing, releasing, operating, and maintenance, no CDR defect was detected.

Evaluation 3 suggests that non-expert reviewers can detect CDR defects in goal-oriented software design reviews. For example, in subsystem 2b, a defect was detected in the design review: ''The definition of a port-level monitoring method for a certain device is omitted.'' The defect could have been detected by goal-oriented check item C2.1.1.1: ''Is the monitoring procedure specified? If one or more descriptions are specified, are they consistent?'' Thus, even if the reviewers are not experts in the system, they might have noticed an omission in the definition of the monitoring procedure.

The results of the case study suggest that the proposed method can potentially detect CDR defects. In the case study, CDR defects were detected with goal-oriented check items. An example of a detected CDR defect is ''Some devices could not communicate with other devices because of incompatible communication protocol versions.'' This defect was detected with the goal-oriented check item C1.1.1: ''Is the communication protocol version compatible between the devices?'' In the requirement definition activity, the requirement explicitly specified the communication protocol name but did not specify the version of the communication protocol.

Sharing a goal tree before goal-oriented software design reviews can reduce the effort for goal-oriented software design reviews. In a discussion with an engineer of the case study, the engineer pointed out that sharing and discussing a goal tree in advance facilitates understanding the corresponding goal-oriented check items and prevents engineers from misunderstanding the specification. He also mentioned that sharing and discussing a goal tree ensure that all reviewers reach a consensus on the specifications before starting the design reviews.

B. THREATS TO VALIDITY

1) INTERNAL VALIDITY

Defining a goal tree and the corresponding goal-oriented check items may require expert-level skills and knowledge in the domain, and personnel overhead. In the case study, the quality assurance engineer who was responsible for the verification of System A defined the goal tree and the corresponding goal-oriented check items. The engineer had general quality-assurance skills but did not have system-specific skills and knowledge and was not a member of the development team. Thus, an engineer with general skills and knowledge of software development and the target system can define a goal tree and the corresponding goal-oriented check items. In addition, the case study showed that the personnel overhead for the proposed method would be small. In the case study, the engineer took 1 hour to identify the goal tree and define the goal-oriented check items for each subsystem 1a and 2a.

If the goal-oriented check items and the check items in the checklist defined in the standard software development process of Company S overlap, the effectiveness of goal-oriented software design reviews will be insufficient. If each goal-oriented check item is included in the standard checklist, reviewers can detect all CDR defects with the standard checklist in standard design reviews. In this case study, the standard design review checklist did not include any goal-oriented check items because the standard design review check items were more general and comprehensive; they were intended for use in the development of various software in the communication network domain, whereas the goal-oriented check items were system-specific.

2) EXTERNAL VALIDITY

If the target system has various goals, such as in the case of a customer relationship management (CRM) system or enterprise resource planning (ERP) system, the effectiveness of the proposed method might be limited. In this case study, the top goal could be easily identified and defined because the communication system had a simple goal tree. By contrast, the goal tree may be more complex in other systems such as CRM and ERP systems. However, goal-oriented requirements analysis methods are not limited by the types of systems. Once a goal tree has been defined, goal-oriented software design reviews can be performed.

A larger number of subgoals may require a larger effort to define goal-oriented check items and perform goal-oriented software design reviews. An engineer who participated in the case study stated that the subgoals needed to be prioritized in case of a larger number of subgoals. Although the proposed method does not consider the priorities of subgoals, the proposed method can easily incorporate subgoal priority via decision-making methods such as an analytic hierarchy process (AHP) [52].

In an iterative development process including agile development process [53]–[56], design and source code are updated in each iteration. Thus, CDR defects are potentially injected in each iteration. Although design reviews might not be explicitly performed in some iterative development processes, CDR defects are detected in activities such as architectural discussion, testing, and implementation of the test and product codes. The rework effort can be reduced if CDR defects are detected using the essence of the proposed method for architectural discussion, testing, and implementation of the test and product codes. For example, potential CDR defects can be identified in end-of-iteration reviews, one of the recommended practices for constant feedback on technical decisions and customer requirements

in agile development process [53], [57]. Specifically, when a context-dependent requirement is implemented in two or more iterations, the implementation can be inconsistent in the iterations. The potential inconsistencies (potential CDR defects) in subsequent iterations can be identified in the end-of-iteration review of the first iteration, in which the context-dependent requirement is implemented. Further work is required to establish the viability of the proposed method to iterative development processes including agile development process.

To generalize the results of the case study, further evaluations in other systems are needed. Because of the limited analysis effort, we carried out lightweight evaluations for other systems developed in Company S. The results of the lightweight evaluations showed that context-dependent ambiguous requirements injected CDR defects and that the CDR defects were overlooked in design reviews and detected in subsequent testing. The CDR defects include inconsistencies among the unit of distance, the notations of time, and the significant digits of numbers. In the lightweight evaluation, we identified the subgoal ''The speed of the moving object can be calculated from the distance moved and the elapsed time.'' We also defined the goal-oriented check item ''Are the measurement methods of the distance moved and the elapsed time correct?'' These results imply that the proposed method can be applied to other systems.

VI. CONCLUSION

This study proposed a method to detect CDR defects by design reviews using a goal tree created via goal-oriented requirements analysis. CDR defects are caused by inconsistencies among design implementations, which are supported by the same requirement (the same subgoal). First, the proposed method creates a goal tree of the target software via goal-oriented requirements analysis. Second, the proposed method defines goal-oriented check items to detect inconsistencies among implementations that realize the same requirement and examine whether the goal and subgoals are satisfied. Third, reviewers perform goal-oriented software design reviews with the goal-oriented check items.

To evaluate the effectiveness of the proposed method, we conducted a case study. The case study evaluated whether the goal-oriented check items could detect CDR defects. The case study also evaluated whether the effort to create a goal tree, define the goal-oriented check items, and perform goal-oriented software design reviews was smaller than the estimated rework effort if the detected defects were overlooked in design review and corrected in subsequent testing. The estimated saved rework effort was calculated as the difference between the sum of the estimated effort for investigating, fixing, and regression testing and the sum of the effort for fixing defects detected in goal-oriented software design reviews assuming that the defect was overlooked in the goal-oriented software design review and detected in subsequent testing. The results of the case study showed that the proposed method detected CDR defects and that other CDR defects were not detected in subsequent testing. The results also showed that the estimated savings in additional rework effort for defects detected by the proposed method was larger than the sum of the effort for preparing and performing the proposed method. Furthermore, the case study investigated whether CDR defects were detected by design reviews without the proposed method and subsequent testing in other subsystems sharing the same goal tree of the target subsystems. The results showed that CDR defects were detected in the other subsystems.

As shown in the case study, we can evaluate the effectiveness percentage of the proposed method by comparing the number of CDR defects detected in goal-oriented software design reviews with the number of CDR defects detected in subsequent testing. More broadly, the effectiveness percentage can be defined as the percentage of CDR defects detected by the proposed method among the CDR defects detected in the entire software development lifecycle.

REFERENCES

[1] A. Porter and L. Votta, ''Comparing detection methods for software requirements inspections: A replication using professional subjects,'' *Empirical Softw. Eng.*, vol. 3, no. 4, pp. 355–379, 1998.
[2] D. M. Berry and E. Kamsties, ''Ambiguity in requirements specification,'' in *Perspectives on Software Requirements*. Norwell, MA, USA: Kluwer, 2004, pp. 7–44.
[3] M. E. Fagan, ''Design and code inspections to reduce errors in program development,'' *IBM Syst. J.*, vol. 15, no. 3, pp. 182–211, 1976.
[4] B. Dhanalaxmi, G. A. Naidu, and K. Anuradha, ''A fault prediction approach based on the probabilistic model for improvising software inspection,'' *Indian J. Sci. Technol.*, vol. 9, no. 45, pp. 1–9, Dec. 2016.
[5] H. Potter, M. Schots, L. Duboc, and V. Werneck, ''InspectorX: A game for software inspection training and learning,'' in *Proc. IEEE 27th Conf. Softw. Eng. Educ. Training (CSEE&T)*, Klagenfurt, Austria, Apr. 2014, pp. 55–64.
[6] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgård, and M. V. Zelkowitz, ''The empirical investigation of perspective-based reading,'' *Empirical Softw. Eng. J.*, vol. 1, no. 2, pp. 133–164, 1996.
[7] A. Van Lamsweerde and E. Letier, ''Integrating obstacles in goal-driven requirements engineering,'' in *Proc. 20th Int. Conf. Softw. Eng.*, Kyoto, Japan, 1998, pp. 53–62.
[8] A. van Lamsweerde, ''Goal-oriented requirements engineering: A guided tour,'' in *Proc. 5th IEEE Int. Symp. Requirements Eng.*, Toronto, ON, Canada, Aug. 2001, pp. 249–262.
[9] A. Lapouchnian, ''Goal-oriented requirements engineering: An overview of the current research,'' Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Tech. Rep., Jun. 2005, vol. 32.
[10] M. Ciolkowski, O. Laitenberger, and S. Biffl, ''Software reviews, the state of the practice,'' *IEEE Softw.*, vol. 20, no. 6, pp. 46–51, Nov./Dec. 2003.
[11] A. A. Porter, L. G. Votta, and V. R. Basili, ''Comparing detection methods for software requirements inspections: A replicated experiment,'' *IEEE Trans. Softw. Eng.*, vol. 21, no. 6, pp. 563–575, Jun. 1995.
[12] F. Shull, I. Rus, and V. Basili, ''How perspective-based reading can improve requirements inspections,'' *Computer*, vol. 33, no. 7, pp. 73–79, Jul. 2000.
[13] T. Thelin, P. Runeson, and B. Regnell, ''Usage-based reading—An experiment to guide reviewers with use cases,'' *Inf. Softw. Technol.*, vol. 43, no. 15, pp. 925–938, Dec. 2001.
[14] T. Thelin, P. Runeson, and C. Wohlin, ''An experimental comparison of usage-based and checklist-based reading,'' *IEEE Trans. Softw. Eng.*, vol. 29, no. 8, pp. 687–704, Aug. 2003.
[15] S. A. Ebad, ''Inspection reading techniques applied to software artifacts—A systematic review,'' *Comput. Syst. Sci. Eng.*, vol. 32, no. 3, pp. 213–226, 2017.
[16] B. P. de Souza, R. C. Motta, and G. H. Travassos, ''The first version of SCENARIotCHECK,'' in *Proc. XXXIII Brazilian Symp. Softw. Eng.*, Salvador, Brazil, Sep. 2019, pp. 219–223.

[17] B. P. de Souza, R. C. Motta, D. D. O. Costa, and G. H. Travassos, "An IoT-based scenario description inspection technique," in *Proc. XVIII Brazilian Symp. Softw. Quality*, Fortaleza, Brazil, Oct. 2019, pp. 20–29.
[18] G. Travassos, F. Shull, M. Fredericks, and V. R. Basili, "Detecting defects in object-oriented designs: Using reading techniques to increase software quality," *ACM SIGPLAN Notices*, vol. 34, no. 10, pp. 47–56, 1999.
[19] O. Laitenberger, "Cost-effective detection of software defects through perspective-based inspections," *Empirical Softw. Eng.*, vol. 6, no. 1, pp. 81–84, 2001.
[20] F. Shull, "Developing techniques for using software documents: A series of empirical studies," M.S. thesis, Dept. Comput. Sci., Univ. Maryland, College Park, MD, USA, 1998.
[21] A. I. Anton, "Goal-based requirements analysis," in *Proc. 2nd Int. Conf. Requirements Eng.*, 1996, pp. 136–144.
[22] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. New York, NY, USA: Wiley, 2009.
[23] A. van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," in *Proc. 26th Int. Conf. Softw. Eng.*, Edinburgh, U.K., 2004, pp. 148–157.
[24] A. van Lamsweerde, R. Darimont, and P. Massonet, "Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt," in *Proc. IEEE Int. Symp. Requirements Eng.*, York, U.K., Mar. 1995, pp. 194–203.
[25] S. Tueno, R. Laleau, A. Mammar, and M. Frappier, "Integrating domain modeling within a formal requirements engineering method," in *Implicit and Explicit Semantics Integration in Proof-Based Developments of Discrete Systems*. Singapore: Springer, 2021, pp. 39–58.
[26] C. Kartiko, A. C. Wardhana, and W. A. Saputra, "Requirements engineering of village innovation application using goal-oriented requirements engineering (GORE)," *Jurnal Infotel*, vol. 13, no. 2, pp. 38–46, May 2021.
[27] X. Franch, L. López, C. Cares, and D. Colomer, "The *i** framework for goal-oriented modeling," in *Domain-Specific Conceptual Modeling*. New York, NY, USA: Springer, 2016, pp. 485–506.
[28] E. S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," in *Proc. 3rd IEEE Int. Symp. Requirements Eng. (ISRE)*, Annapolis, MD, USA, Jan. 1997, pp. 226–235.
[29] E. S. K. Yu, "Modeling organizations for information systems requirements engineering," in *Proc. IEEE Int. Symp. Requirements Eng.*, San Diego, CA, USA, Jan. 1993, pp. 34–41.
[30] Y. Wang, T. Li, Q. Zhou, and J. Du, "Toward practical adoption of i* framework: An automatic two-level layout approach," *Requirements Eng.*, vol. 26, no. 3, pp. 301–323, Sep. 2021.
[31] A. S. Vingerhoets, S. Heng, and Y. Wautelet, "Using i* and UML for blockchain oriented software engineering: Strengths, weaknesses, lacks and complementarity," *Complex Syst. Informat. Model. Quart.*, vol. 149, no. 26, pp. 26–45, 2021.
[32] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Trans. Softw. Eng.*, vol. 18, no. 6, pp. 483–497, Jun. 1992.
[33] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*. Berlin, Germany: Springer, 2012.
[34] J. Mylopoulos, L. Chung, and E. Yu, "From object-oriented to goal-oriented requirements analysis," *Commun. ACM*, vol. 42, no. 1, pp. 31–37, Jan. 1999.
[35] S. S. Paradkar, "A framework for modeling non-functional requirements for business-critical systems," *Int. J. Innov. Res. Comput. Sci. Technol.*, vol. 9, no. 1, pp. 15–19, Jan. 2021.
[36] J. Cleland-Huang, O. Gotel, and A. Zisman, *Software and Systems Traceability*. New York, NY, USA: Springer, 2012.
[37] O. C. Z. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem," in *Proc. IEEE Int. Conf. Requirements Eng.*, Colorado Springs, CO, USA, Apr. 1994, pp. 94–101.
[38] G. Spanoudakis and A. Zisman, "Software traceability: A roadmap," in *Handbook of Software Engineering and Knowledge Engineering*. Singapore: World Scientific, 2005, pp. 395–428.
[39] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, "Requirements traceability: A systematic review and industry case study," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 22, no. 3, pp. 385–433, May 2012.
[40] S. Nair, J. L. De La Vara, and S. Sen, "A review of traceability research at the requirements engineering conference," in *Proc. 21st IEEE Int. Requirements Eng. Conf. (RE)*, Rio de Janeiro, Brazil, Jul. 2013, pp. 222–229.
[41] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained BERT models," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, Madrid, Spain, May 2021, pp. 324–335.
[42] A. Zisman and A. Kozlenkov, "Knowledge base approach to consistency management of UML specifications," in *Proc. 16th Annu. Int. Conf. Automated Softw. Eng. (ASE)*, San Diego, CA, USA, 2001, pp. 359–363.
[43] A. Kozlenkov and A. Zisman, "Are their design specifications consistent with our requirements?" in *Proc. IEEE Joint Int. Conf. Requirements Eng.*, Essen, Germany, Sep. 2002, pp. 145–154.
[44] J. Hassine, J. Rilling, and J. Hewitt, "Change impact analysis for requirement evolution using use case maps," in *Proc. 8th Int. Workshop Princ. Softw. Evol. (IWPSE)*, Lisbon, Portugal, 2005, pp. 81–90.
[45] S. Lehnert, "A review of software change impact analysis," Dept. Softw. Syst./Process Inform., Ilmenau Univ. Technol., Ilmenau, Germany, Tech. Rep., 2011.
[46] T. Jalaja, T. Adilakshmi, and P. S. R. Abhishek, "Automation of change impact analysis for Python applications," in *Smart Computing Techniques and Applications*. New York, NY, USA: Springer, 2021, pp. 259–267.
[47] S. B. R. Arnold, *Software Change Impact Analysis* (Tutorial Series). Los Alamitos, CA, USA: Wiley, 1996.
[48] S. A. Bohner, "Impact analysis in the software change process: A year 2000 perspective," in *Proc. Int. Conf. Softw. Maintenance (ICSM)*, Monterey, CA, USA, 1996, pp. 42–51.
[49] B. G. Ryder and F. Tip, "Change impact analysis for object-oriented programs," in *Proc. ACM SIGPLAN-SIGSOFT Workshop Program Anal. Softw. Tools Eng. (PASTE)*, Snowbird, UT, USA, 2001, pp. 46–53.
[50] *Automotive SPICE Process Assessment and Reference Model Version 3.1*, Qual. Manage. Center, Sect. German Automot. Ind. Assoc., Berlin, Germany, 2017.
[51] *CMMI for Development, Version 1.3*, Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2010.
[52] T. L. Saaty, *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*. New York, NY, USA: McGraw-Hill, 1980.
[53] J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," *Computer*, vol. 34, no. 9, pp. 120–127, 2001.
[54] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New directions on agile methods: A comparative analysis," in *Proc. 25th Int. Conf. Softw. Eng.*, Portland, OR, USA, 2003, pp. 244–254.
[55] A. L. Lorca, R. Burrows, and L. Sterling, "Teaching motivational models in agile requirements engineering," in *Proc. IEEE 8th Int. Workshop Requirements Eng. Educ. Training (REET)*, Banff, AB, Canada, Aug. 2018, pp. 30–39.
[56] N. Potter and M. Sakry, "Implementing SCRUM (Agile) and CMMI together," *Process Group-Post Newslett.*, vol. 16, no. 2, pp. 1–6, 2009.
[57] E. Rubin and H. Rubin, "Supporting agile software development through active documentation," *Requirements Eng.*, vol. 16, no. 2, pp. 117–132, Jun. 2011.

MICHIYO WAKIMOTO received the master's degree in management of technology from Ritsumeikan University, Japan. She is currently pursuing the Ph.D. degree with the Graduate School of Informatics, Nagoya University, Japan. Her research interests include empirical software engineering and software quality.

SHUJI MORISAKI (Member, IEEE) received the D.E. degree in information science from the Nara Institute of Science and Technology, Japan, in 2001. He is currently an Associate Professor. Previously, he was a Software Engineer with Japanese Software Industry. His research interests include empirical software engineering and software quality.