An Empirical Evaluation of the Effectiveness of Inspection Scenarios Developed from a Defect Repository

Kiyotaka Kasubuchi
Research and Development Center
SCREEN Holdings Co., LTD
Fushimi, Kyoto, Japan

Shuji Morisaki
Graduate School of Information Science
Nagoya University
Nagoya, Aichi, Japan

Akiko Yoshida
Research and Development Center
SCREEN Holdings Co., LTD
Fushimi, Kyoto, Japan

Chikako Ogawa
Faculty of Informatics,
Shizuoka University
Johoku, Hamamatsu,
Shizuoka, Japan

***Abstract*—Abstracting and summarizing high-severity defects detected during inspections of previous software versions could lead to effective inspection scenarios in a subsequent version in software maintenance and evolution. We conducted an empirical evaluation of 456 defects detected from the requirement specification inspections conducted during the development of industrial software. The defects were collected from an earlier version, which included 59 high-severity defects, and from a later version, which included 48 high-severity defects. The results of the evaluation showed that nine defect types and their corresponding inspection scenarios were obtained by abstracting and summarizing 45 defects in the earlier version. The results of the evaluation also showed that 46 of the high-severity defects in the later version could be potentially detected using the obtained inspection scenarios. The study also investigated which inspection scenarios can be obtained by the checklist proposed in the value-based review (VBR). It was difficult to obtain five of the inspection scenarios using the VBR checklist. Furthermore, to investigate the effectiveness of cluster analysis for inspection scenario development, the 59 high-severity defects in the earlier version were clustered into similar defect groups by a clustering algorithm. The results indicated that cluster analysis can be a guide for selecting similar defects and help in the tasks of abstracting and summarizing defects.**

***Index Terms*—Software inspection, defect abstraction, prioritizing inspection scenarios**

I. Introduction

Many techniques have been proposed to increase development efficiency and improve software quality. Software inspection is one such technique that is aimed at detecting defects in the early stages of software development [7]. The use of an inspection scenario or a checklist allows inspectors to focus on specific defect types. Studies of inspection scenario have found that focusing on specific defect types results in efficient and effective inspection [1][6][14][15].

As software grows in size, comprehensive defect detection during inspection becomes increasingly difficult because of the limited inspection time and resources. One possible solution is to prioritize inspection scenarios or questions in a checklist and start the inspection in the order of highest to lowest priority, in this way the detection effort focuses initially on high-severity defects such as those that might halt business operations.

Usage-based reading (UBR) [17] and value-based review (VBR) [11] are techniques that prioritize inspection scenarios. UBR defines inspection scenarios and their priorities from use cases prioritized by the importance level before defect detection is initiated. In a UBR, inspectors start defect detection with the highest priority scenario. In a VBR, inspectors assign a higher detection priority to defects categorized as defect types that have the potential to spoil the higher value capabilities of the target software. The stakeholders of the software determine the order of priority using a checklist to identify the higher value capabilities. The checklist consists of five categories; completeness, consistency/feasibility, ambiguity, conformance, and risk. In a VBR, inspectors start defect detection with the highest defect types.

During software maintenance and evolution, the inspection scenarios and questions in a checklist are expected to be prioritized according to the defects detected in previous software versions. However, to the best of our knowledge, there is no published study prioritizing inspection scenarios, or checklist questions in this way. This paper investigates whether inspection scenarios developed from the high-severity defects detected in the previous software versions can detect high-severity defects in a subsequent version and whether the inspection scenarios can be developed by another approach. Also, an investigation whether computer-supported analysis guides developing inspection scenarios will be conducted. More specifically, the study was conducted to answer the following research questions:

RQ1: How effective is developing inspection scenarios S_h from high-severity defects detected during inspections of previous software versions?

RQ1-1: Can the inspection scenarios S_h detect a larger number of high-severity defects than detected by inspection scenarios S_m and S_l, which were developed from medium- and low-severity defects?

RQ1-2: Can the inspection scenarios S_h be developed by the checklist proposed in the VBR?

RQ2: Can computer-supported analysis of the defects detected in the inspections of previous software versions guide the development of inspection scenarios?

The study used 456 defects detected during the inspections of requirement specifications of two versions of the same soft-

TABLE I. EXAMPLES OF DEFECTS

Defect ID	Severity	Description	…
d_1	High	Order is accepted without specifying quantity purchase.	…
d_2	High	Order is accepted without specifying shipping address.	…
d_3	Medium	Despite a free shipping campaign, a shipping charge appears on the shopping cart screen.	…
…	…	…	…

TABLE II. EXAMPLES OF DEFECT TYPES AND INSPECTION SCENARIOS

Defect type ID	Defect ID	Description	Scenario ID	Inspection Scenario	…
T_1	d_1, d_2	Order is accepted without required information.	S_1	Are input validations defined when order is accepted?	…
…	…	…			…

ware in an industry. First, an engineer who has knowledge and experience with the development of the software identified defect types and developed inspection scenarios based on the high-severity defects detected in the earlier version. Then, two analysts determined whether the defects detected in the later version were detectable with the inspection scenarios. Finally, we generated clusters of defects with an algorithm to compare identified defect types.

The rest of the paper is organized as follows. Section 2 introduces related research. Section 3 provides an overview of the development of inspection scenarios developed from defects detected in previous software versions. The settings and the results of the evaluation are described in Sections 4 and 5. Section 6 discusses the evaluation results. Section 7 concludes the study.

II. RELATED RESEARCH

A number of approaches for developing scenarios to use in a software inspection have been proposed. An inspection scenario is a guide for an inspector in defect detection. Perspective-based reading (PBR) defines inspection scenarios from the point-of-view of the stakeholders such as a user or a programmer [2]. Inspection scenarios used in a PBR can be developed to detect high-severity defects. However, PBR does not refer to the defects detected in previous software versions.

Thelin et al. proposed UBR. Inspection scenarios used in a UBR are defined according to the use cases of the software [17]. Inspection scenarios are prioritized according to the importance of the corresponding use case. In a UBR, the use of scenarios can only detect the defects that can be captured by the use cases. Also, important use cases do not necessarily correspond to high-severity defects.

Porter et al. proposed scenario-based reading (SBR). Inspection scenarios used in an SBR are defined according to the defect types that should be detected. In their article [15], three basic defect types were presented: data type consistency, incorrect functionality, and ambiguity or missing functionality. Although inspection scenarios can in fact be developed to detect high-severity defects in previous software versions, the article does not specifically discuss these.

The error abstraction process (EAP) enables inspectors to detect similar defects in re-inspection and fix defects in the same inspection [10][19]. The EAP is conducted after a defect detection process. In an EAP, detected defects are analyzed and abstracted into error taxonomy. In a subsequent re-inspection process, the error taxonomy helps inspectors detect defects similar to those detected in the defect detection process. Although error taxonomy obtained in an EAP could be used in an inspection of a subsequent version, we found no published study that evaluated the effectiveness of the taxonomy in a subsequent version.

Chernak proposed an approach using causal analysis for developing questions in a checklist from the defects detected during inspections [4]. Although these defects can be a set of high-severity defects, the article does not specifically mention this.

Shihab et al. investigated post-release defects to predict high-impact defects in source code from process execution history and source code metrics [16]. In the study, the prediction model predicted defect-prone source code modules. The procedures to identify defect types or the inspection scenarios from the prediction model were not described.

III. INSPECTION SCENARIOS FROM A DEFECT REPOSITORY

A. Definition

A set of defects $D = \{d_1, d_2, \dots, d_m\}$ in a defect repository is a set of defects detected in software inspections conducted during the development of previous software versions. Table I shows an example of defect repository D. The defects are detected during the inspections of the development of an online shopping system. Each defect d has certain attributes, including a defect description written in natural language and a severity level.

Inspection scenarios $S = \{S_1, S_2, \dots, S_n\}$ are developed from defect types $T = \{T_1, T_2, \dots, T_n\}$. Defect type T_j is identified by abstracting and summarizing one or more defects $\{d_k, d_l, \dots\}$. Table II shows an example of defect types T and corresponding inspection scenarios S. The defect types T are abstracted and summarized from the defects in Table I. Defects d_1 and d_2 indicate that making an order is possible without specifying the quantity purchased or the shipping address. Defect type T_1 is identified from d_1 and d_2. Inspection scenario S_1 will detect defects categorized as defect type T_1 in inspections of a subsequent version.

B. Roles

The roles in the development of inspection scenarios and subsequent inspections are as follows:

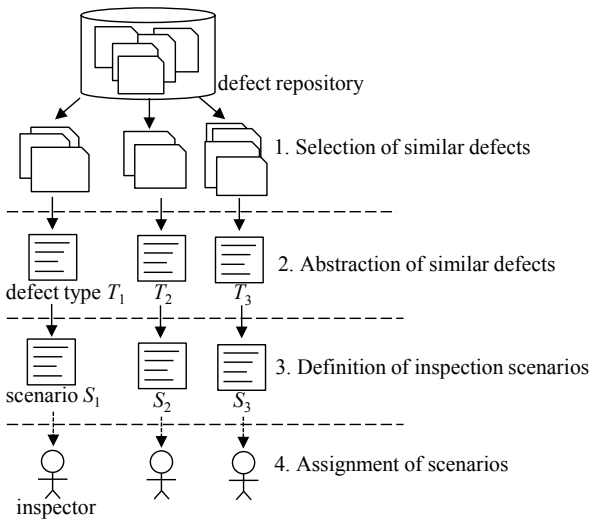

Fig. 1. Overview of the development of inspection scenarios

TABLE III. THE NUMBER OF DEFECTS IN THE EVALUATION

Group	Total	Severity		
		High	***Medium***	***Low***
D_A	249	59(23.7%)	124(49.8%)	66(26.5%)
D_B	207	48(23.2%)	100(48.3%)	59(28.5%)

- The ***analyst*** identifies defect types and develops inspection scenarios. The analyst has knowledge of the software that is to be inspected, its domain, and its typical and frequent defects.
- The ***moderator*** assigns inspection scenarios to inspectors. The moderator leads inspections and has knowledge of the skill and experience of the inspectors.
- The ***inspector*** detects defects according to the assigned inspection scenario.

C. Procedure

Figure 1 shows an overview of the development of an inspection scenario from the defects in the defect repository. The procedure is as follows:

1. *Selection of similar defects*: The analyst selects similar defects from the defect repository, making use of the keywords in the defect descriptions.
2. *Abstraction and summarization of similar defects into defect types*: The analyst summarizes similar defects by abstracting them into defect types *T*.
3. *Definition of inspection scenarios corresponding to defect types*: The analyst defines a set of inspection scenarios *S*. Each inspection scenario $S_i \in S$ enables inspectors to detect defects categorized as a defect type $T_i \in T$.
4. *Assignment of scenarios to inspectors:* The moderator assigns inspection scenarios to inspectors.
5. *Detection of defects according to scenarios:* Inspectors detect defects using the inspection scenarios.

IV. EVALUATION

A. Defect Repository

The defect repository for the evaluation was collected from the requirement specification inspection conducted during the development of two versions of a commercial manufacturing system. Documented inspection scenarios or checklists were not used in the inspections of both versions. The duration of defect collection was nine months. The requirement specification document and detected defect descriptions were written in Japanese. The user interface and the manual of the system were provided in Japanese and English versions.

The defects detected in the earlier version were defects D_A. The defects detected in the later version were defects D_B. Defects D_A and D_B comprised all the defects detected in every requirement inspection in two versions. Table III shows the distribution of defect severity.

Each defect in the defect repository included the following information:

- The version in which it was detected
- The date it was detected: the date of the inspection meeting
- Its severity: the degree of damage to a user (high, medium, or low)
- A detailed description of the defect, question, or concerns in natural language

B. Procedure

1) Effectiveness of the Inspection Scenarios (RQ1): Analyst X identifies defect types in T_h, T_m, and T_l from defects D_{Ah}, D_{Am}, and D_{Al}, respectively. Defects D_{Ah}, D_{Am}, and D_{Al} are high-, medium-, and low-severity defects in D_A. The analyst then tries to develop inspection scenarios S_h, S_m, and S_l corresponding to the defect types T_h, T_m, and T_l. To reduce the dependency on one analyst, after analyst X identifies the defect types and develops the inspection scenarios, analyst Y examines whether the defect types and the inspection scenarios are adequate. If analyst Y judges that a defect type and/or an inspection scenario are not adequate, analyst Y asks analyst X to review and change the defect type and/or the inspection scenario.

Analyst X determines whether each defect in D_B could be detected by inspection scenarios S_h, S_m, and S_l. Materials given to the analysts include inspection scenarios, defects D_B, and the requirements specification. Each determination is conducted virtually without re-inspection due to limited resources. The analysts assume the inspection was conducted by an inspector with typical skill and knowledge in the development project, and then they determine whether the inspector detected each defect in D_B.

Also, analyst Y evaluates whether the inspection scenarios can be obtained from another approach. We selected the VBR as another approach to develop inspection scenarios because the VBR prioritizes inspection scenarios with defect criticality and category. Defect criticality rates defects as high-criticality, medium-criticality, or low-criticality. Defect category consists

of completeness, consistency/feasibility, ambiguity, conformance, and risk. For example, a defect category "critical missing elements: backup/recovery, external interfaces, success-critical stakeholders, critical exception handling, missing priorities" is categorized as belonging to the completeness category and as having high-criticality. Twenty four defect types are proposed in the VBR.

A detailed procedure is as follows:

1. Extracting words from the defect description: words w_k and corresponding frequency f_k are extracted from the "defect description" in all defect descriptions in D_{Ah} ($f_k \geq f_{k+1}$).
2. Choosing defects by w_k: subset of defects $D_{Ahk} = \{d_{k1}, d_{k2}, \ldots, d_{kp}\}$ is chosen for each w_k, where the defect description of d_{ki} includes the word w_k, $1 \leq k \leq 20$, and $D_{Ah1} \wedge D_{Ah2} \wedge \ldots = \varnothing$.
3. Abstracting and summarizing defects in D_{Ahk}: Analyst X attempts to abstract and summarize defects in D_{Ahk} or a subset of D_{Ahk} into defect types T_h.
4. Developing inspection scenarios S_h: Analyst X attempts to develop inspection scenario $S_{hi} \in S_h$ that can detect defects categorized as defect type $T_{hi} \in T_h$.
5. Examining the defect types and inspection scenarios: Analyst Y examines whether each identified defect type in T_h and each developed inspection scenario in S_h are adequate. If needed, analyst Y asks analyst X to review and change the defect type and/or the inspection scenario.
6. Developing and examining inspection scenarios S_m and S_l by the same way as developing the inspection scenarios S_h: Analyst X develops inspection scenario S_m and S_l by identified defect types T_m and T_l. Defect types T_m and T_l are abstracted and summarized from defect D_{Am} and D_{Al}, respectively. Analyst Y examines whether each identified defect type in T_m and T_l and each inspection scenario in S_m and S_l are adequate. If needed, analyst Y asks analyst X to review and change the defect type and/or the inspection scenario.
7. Determining whether defects D_B can be detected by inspection scenarios S_h, S_m, and S_l (RQ1-1): Analyst X determines whether each defect in D_B could be potentially detected using the inspection scenarios S_h, S_m, and S_l separately. Analyst Y ensures that each determination is adequate. If needed, analysts Y asks analyst X to review and change the determination. Overlaps are allowed among the defects that are determined to have the potential to be detected by S_h, S_m, and S_l.
8. Comparing with another approach (RQ1-2): Analyst Y examines whether inspection scenarios S_h can be obtained from the checklist proposed in the VBR. Each question in the checklist has one of the three criticality levels. In the evaluation, all questions are used.

2) Computer-Supported Analysis (RQ2): To evaluate that clustering the defects detected in previous software versions supports the identification of defect types, defects D_{Ah} were clustered into similar subsets using the Ward clustering method [18], which is a basic clustering algorithm. We chose a basic clustering algorithm as a first trial. A detailed procedure

TABLE IV. DEFECT TYPES AND INSPECTION SCENARIOS IDENTIFIED FROM HIGH-SEVERITY DEFECTS D_{AH}

Defect Type	Inspection Scenario	# of Defects Abstracted
T_{h1} Lack of strict timing in the definition for *counting* the number of objects in production	S_{h1} Are the definitions of the strict timings of *counting* objects clear enough for programmers to identify the sequence of *counting*? Ensure that the definitions of *counting process* include a timing chart.	9
T_{h2} Lack of consideration of changes from the previous versions	S_{h2} Are unexpected side-effects of changes in reused component of previous versions are discussed and considered? Ensure that no changes exist in series of events, message formats in the communication protocol, and access privilege.	9
T_{h3} Lack of information in a sequence of screens	S_{h3} Is information on each screen strictly defined? Is there no omitted information in a sequence of screens? Check the series of screens and verify omitted and incorrect input sequence according to use cases.	7
T_{h4} Ambiguous definitions of *control*	S_{h4} Are the definitions of the manufacturing *controls* and *processes* clear enough for programmers to implement? Ensure that the sequence, pre-conditions, and post-conditions of *controls* and *processes* are defined.	4
T_{h5} Inappropriate schema definition	S_{h5} Are *counter* values for monitoring expected to be of variable length or fixed length? Check whether the variables have a fixed or variable length.	4
T_{h6} Ambiguous or missing definition of validation on *process P*	S_{h6} Are validation items and their criteria strictly defined for *process P*? Ensure that the definitions exist in the description of *process P*.	3
T_{h7} Lack of input data and procedure definitions of *converting*	S_{h7} Are input values and pre-conditions for *converting* defined? Ensure that the given variables for *converting* are defined and that pre-conditions of *converting* are defined.	3
T_{h8} Ambiguous or missing definitions of input values for *counting* capabilities	S_{h8} Do interface definitions accessing *counting* capabilities have strict definitions of input values?	3
T_{h9} Lack of definition of input file format	S_{h9} Are file formats of input files defined? Enumerate the input files and ensure that the formats of input files are defined in the appendix.	3

TABLE V. FREQUENTLY USED WORDS IN DEFECT DESCRIPTIONS OF D_{AH}

Rank	Word	Frequency
1	count	37
2	information	25
3	event	21
3	count value	21
3	mandate	21
6	feature α	14
6	model X	14
6	alarm	14
10	area	13

is as follows:

1. Clustering defects: The defects D_{Ah} are clustered into k clusters $C = \{C_{1, ..., } C_k\}$ using the Ward clustering method. Clustering is conducted with the defect descriptions written in natural language. The Ward clustering method requires that the number of clusters be given. The numbers of clusters k are 5, 10, and 15 because we assumed that the numbers of inspection scenarios varied between one and three, and that the number of inspectors was five.
2. Identifying overlap between the obtained defect types T_h and the clusters: Analyst X compares each cluster and defect types T_h and evaluates the coverage of the defect types T_h by the clusters obtained. Analyst Y ensures that each evaluation is adequate. If needed, analyst Y can ask analyst X to review and change the evaluation.
3. Evaluating correspondence between the characteristic words of each cluster and defect type: Analyst X evaluates that the characteristic words in each cluster represent a defect type in T_h. The characteristic words are those that have the largest Jaccard similarity coefficient [9] in each cluster. The Jaccard similarity coefficient $j(C_i, w)$ is obtained for each word w as follows: $j(C_i, w) = | D_w \wedge C_i | / | D_w \vee C_i |$ where C_i is a cluster and D_w is a set of defects whose defect description contained word w.

V. RESULT

A. Effectiveness of the Inspection Scenarios (RQ1)

The defect types and inspection scenarios are shown in Table IV. In Table IV, words in italics such as process, count, control, and convert are specific words used in the manufacturing system. For example, the word "process" does not refer to a general computational process but to a manufacturing process. Defect type T_{h2}, "Lack of consideration of changes from the previous software versions" was not added by the analysts, but was summarized from defect descriptions such as "changes from version 2.x are not considered."

Table V shows words w_k extracted from the defect descriptions of defects D_{Ah}. The defects were originally written in Jap-

TABLE VI. DEFECT TYPES AND INSPECTION SCENARIOS IDENTIFIED FROM MIDDLE-SEVERITY DEFECTS D_{AM}

Defect Type	Inspection Scenario	# of Defects Abstracted
T_{m1} Omitted execution precondition and omitted parameter validation before manufacturing *process* executions	S_{m1} Are pre-conditions and parameter validations defined appropriately before manufacturing *process* executions? Ensure that the pre-conditions and parameter validations are fully specified.	13
T_{m2} Omitted specification on GUI component statuses	S_{m2} Are the availabilities of the GUI components and the events/triggers for changing availabilities defined? Check the possibility of changing GUI component statuses and the events/triggers in the user interface specification document. Ensure that the definitions exist.	12
T_{m3} Ambiguous or incorrect range limitations	S_{m3} Are the ranges of transition time, pressure, and rotation speed in manufacturing *processes* defined and valid? Ensure that the range limitations of the *process* executions are satisfied by referring to corresponding hardware specifications and *process* definition files.	11
T_{m4} Ambiguous or omitted layout definition of GUI components and messages	S_{m4} Are the detailed layout of GUI components and messages for users defined? Ensure that neither ambiguous layout definitions nor undefined messages exist in the user interface specification document.	11
T_{m5} Ambiguous, incorrect, or omitted explanations in screen messages	S_{m5} Do the messages cause user misunderstanding? Ensure that no misleading messages exist in the screen messages. If the messages are changed by events/triggers, ensure that events/triggers are strictly defined and that there is no inconsistency between events/triggers and message transitions.	9
T_{m6} Insufficient explanation of what *process* is to be executed by each GUI buttons	S_{m6} Are the mappings of GUI buttons to *process* execution defined? Ensure that each GUI button has the definition of triggered *process* execution.	6
T_{m7} Insufficient consideration of queue overflow	S_{m7} Are exception handlings for queue overflow defined if the persistent data store uses queues? Find the capabilities of the persistent data store in the specification, and check whether the capabilities use queues. Ensure that exception handlings for queue overflow are defined, if queue is used.	4
T_{m8} Lack of language settings for user interface	S_{m8} Is the language for the user interface defined? If two or more languages can be selected, are the default language and language settings defined?	4
T_{m9} Ambiguous definition of dialog box (modal window) and messages on it	S_{m9} Are events/triggers for showing dialog boxes defined in the specification? Are the screen locations of the dialog boxes and the messages in the dialog boxes defined? Find dialog boxes in the user interface specification and ensure that the events/triggers for showing dialog boxes, displaying location, and messages in the dialog boxes are defined.	3

TABLE VII. DEFECT TYPES AND INSPECTION SCENARIOS IDENTIFIED FROM LOW-SEVERITY DEFECTS D_{AL}

Defect Type	Inspection Scenario	# of Defects Abstracted
T_{l1} Misleading messages on GUI component and inconsistency among labels of GUI component	S_{l1} Are there misleading messages on GUI components? For example, alert messages do not disappear even though the alerting conditions are no longer satisfied. Are there consistencies among similar GUI components including groups of labels and positions of GUI components?	12
T_{l2} Lack of referred document	S_{l2} Are documents, figures and tables that are referred to available? Find external references in the specification and ensure that the external references are available.	4
T_{l3} Incorrect screen transition chart	S_{l3} Are screen transitions correctly specified in the screen transition chart? Ensure that each transition and transition condition is described as expected and intended.	2
T_{l4} Lack of information indicating *process* completion	S_{l4} Do the criteria or exit status indicate that process completed with a normal status as described? Ensure that notification, status, or information indicating process completion is described.	2
T_{l5} Insufficient explanations of the mapping of user operations to commands for the manufacturing subsystem	S_{l5} Are the definitions of the user's operations and the corresponding executed commands for the manufacturing subsystem clearly described? If a user operation may execute two or more commands for the subsystems, is condition for each execution clear?	2

TABLE VIII. FREQUENTLY USED WORDS IN DEFECT DESCRIPTIONS OF D_{AM}

Rank	Word	Frequency
1	display	61
2	recipe	58
3	in case of	57
4	button	55
5	folder	47
6	control command A	35
7	queue	29
8	validation	27
9	mm (millimeter)	26
10	less than	24
10	screen	24

TABLE IX. FREQUENTLY USED WORDS IN DEFECT DESCRIPTIONS OF D_{AL}

Rank	Word	Frequency
1	button	32
2	indication	29
3	implementation items	21
3	screen	21
3	control command B	21
6	label	14
6	area	14
6	in case of	14
6	alarm	14
10	area	13

anese and words in Table V were originally single words. Some words in Table V consist of two words because the words were translated into English. The frequency is not the number of defects d_k but the number of word appearances in the defect descriptions of defects D_{Ah}. The most frequently used word in defect descriptions in defects D_{Ah} was "count," with 37 appearances. The number of defects whose defect descriptions included the word "count" was 18. The analysts selected nine defects that described ambiguities in the timing of counting the number of objects in production in the manufacturing system. The defects were abstracted and summarized as the defect type "lack of strict timing in the definition for counting the number of objects in production." The corresponding

TABLE X. THE DISTRIBUTIONS OF DEFECTS THAT CAN POTENTIALLY BE DETECTED USING INSPECTION SCENARIOS S_H

Scenario	Total	Severity		
		High	*Medium*	*Low*
S_{h1}	10	10	0	0
S_{h2}	42	10	22	10
S_{h3}	30	4	12	14
S_{h4}	4	4	0	0
S_{h5}	12	5	6	1
S_{h6}	4	4	0	0
S_{h7}	3	3	0	0
S_{h8}	6	3	3	0
S_{h9}	7	3	2	2
All	118	46	45	27

TABLE XI. THE DISTRIBUTIONS OF DEFECTS THAT CAN POTENTIALLY BE DETECTED USING INSPECTION SCENARIOS S_M

Scenario	Total	Severity		
		High	*Medium*	*Low*
S_{m1}	4	0	4	0
S_{m2}	20	0	12	8
S_{m3}	9	1	8	0
S_{m4}	31	3	8	20
S_{m5}	18	1	11	6
S_{m6}	7	0	5	2
S_{m7}	9	1	4	4
S_{m8}	3	0	3	0
S_{m9}	1	0	1	0
All	102	6	56	40

inspection scenario was defined as "Are the definitions of the strict timings of counting objects clear enough for programmers to identify the counting sequence?"

The analysts identified defect types T_m and T_l and inspection scenarios S_m and S_l. The defect types and inspection scenarios are shown in Table VI and VII. Table VIII shows words w_k extracted from D_{Am}. Table IX shows words w_k extracted from D_{Al}.

TABLE XII. THE DISTRIBUTIONS OF DEFECTS THAT CAN POTENTIALLY BE DETECTED USING INSPECTION SCENARIOS S_L

Scenario	Total	Severity		
		High	*Medium*	*Low*
S_{l1}	7	0	7	0
S_{l2}	1	0	1	0
S_{l3}	0	0	0	0
S_{l4}	2	0	0	2
S_{l5}	0	0	0	0
All	10	0	8	2

TABLE XIII. COVERAGE BY THE CLUSTERS FROM D_{AH}

Defect type	Number of Defects	Corresponding Cluster		
		C_a	C_b	C_c
T_{h1}	9	C_{a1}	C_{b1}	C_{c1}
T_{h2}	6	C_{a2}	C_{b2}	C_{c2}
	1	C_{a3}	C_{b3}	C_{c3}
	1	C_{a4}	C_{b4}	C_{c4}
	1	C_{a5}		
T_{h3}	2	C_{a6}	C_{b5}	
	3	C_{a7}	C_{b6}	C_{c3}
	2	C_{a8}	C_{b3}	
T_{h4}	2	C_{a3}		
	2	C_{a9}	C_{b7}	C_{c5}
T_{h5}	4	C_{a10}	C_{b8}	
T_{h6}	3	C_{a11}	C_{b9}	C_{c4}
T_{h7}	3	C_{a12}	C_{b10}	C_{c5}
T_{h8}	3	C_{a9}	C_{b7}	
T_{h9}	3	C_{a12}	C_{b10}	
Not categorized	1	C_{a9}	C_{b7}	
	1	C_{a6}	C_{b5}	C_{c4}
	1	C_{a4}	C_{b4}	
	3	C_{a5}		
	2	C_{a13}		
	4	C_{a14}		
	2	C_{a15}		

The analysts determined that a subset of defects D_B could be potentially detected using the obtained inspection scenarios S_h, S_m, and S_l. Table X shows the distribution of defects that have the potential to be detected using inspection scenarios S_h. We performed Fisher's exact test on the distributions of defect severity. The null hypothesis, "the distributions of defect severity in D_B and the distribution of defect severity of defects that have the potential to be detected using S_h are the same," was rejected at a significance level of 0.05 (p = 0.012). The percentage of high-severity defects categorized detected using inspection scenarios S_h was 40.0%. This percentage is 1.7 times larger than the percentage of high-severity defects in D_B. The percentage of medium-severity defects detected using S_h was 88.4% larger than the percentage of medium-severity defects in D_B. It was determined that inspection scenarios S_{h1}, S_{h4}, S_{h6}, and S_{h7} detect only the high-severity defects. Here, high-severity defects are the defects categorized as high severity defects by the inspectors when they were stored into the defect repository.

Table XI shows the distributions of defects that have the potential to be detected using inspection scenarios S_m. The null hypothesis, "the distribution of defect severity in D_B and the distribution of defect severity in the defects that have the potential to be detected using S_m are the same," was rejected by Fisher's exact test at a significance level of 0.05 (p = 0.00027). Most of the defects judged to be potentially detected by inspection scenarios S_m were medium- or low-severity defects

Table XII shows the distributions of defects that have the potential to be detected using inspection scenarios S_l. The null hypothesis, "the distribution of defect severity in D_B and the distribution of defect severity that have the potential to be detected by S_l are the same," was not rejected by Fisher's exact test at a significance level of 0.05 (p = 0.15). No defect in D_B was determined to be detectable by inspection scenarios S_{l3} and S_{l5}.

For RQ1-2, analyst Y also determined that four inspection scenarios S_{h3}, S_{h5}, S_{h8}, and S_{h9} could be obtained using the VBR checklist.

TABLE XIV. WORDS WITH A LARGER JACCARD SIMILARITY COEFFICIENT IN THE CLUSTERS C_A

Cluster	Words and Jaccard Similarity Coefficients
C_{a1}	timing(0.89), event(0.75), counter(0.70)
C_{a2}	message(1.00), control(0.86), change(0.55)
C_{a3}	operation(1.00), manual(1.00), figure(0.67)
C_{a4}	recipe(0.67), minimum(0.50), column(0.50)
C_{a5}	file(1.00), foundation(1.00), cell(0.50)
C_{a6}	folder(1.00), unit(0.75), error(0.67)
C_{a7}	present(1.00), link(1.00), module(0.75)
C_{a8}	perspective(1.00), monitor(1.00), two or more(1.00)
C_{a9}	operation(0.75), data(0.60), monitoring(0.50)
C_{a10}	persistence(1.00), schema(0.80), database(0.67)
C_{a11}	lock(1.00), library(0.75), forbidden(0.67)
C_{a12}	procedure(1.00), converting(0.83), xml(0.75)
C_{a13}	instruction(0.50), operation(0.50), manual(0.50)
C_{a14}	subsystem(1.00), brush(1.00), device(0.50)
C_{a15}	production version(0.50),format(0.50),parameter(0.50)

TABLE XV. WORDS WITH A LARGER JACCARD SIMILARITY COEFFICIENT IN THE CLUSTERS C_B

Cluster	Words and Jaccard Similarity Coefficients
C_{b1}	timing(0.89), event(0.75), counter(0.70)
C_{b2}	message(1.00), control(0.86), change(0.55)
C_{b3}	manual(0.60), operation(0.60), all(0.43)
C_{b4}	foundation(0.29), recipe(0.25), parameter(0.21)
C_{b5}	folder(1.00), unit(0.75), error(0.67)
C_{b6}	present(1.00), link(1.00), module(0.75)
C_{b7}	operation(0.75), data(0.60), monitoring(0.50)
C_{b8}	persistence(1.00), schema(0.80), database(0.67)
C_{b9}	lock(1.00), library(0.75), forbidden(0.67)
C_{b10}	procedure(1.00), converting(0.83), xml(0.75)

TABLE XVI. WORDS WITH A LARGER JACCARD SIMILARITY COEFFICIENT IN THE CLUSTERS C_C

Cluster	Words and Jaccard Similarity Coefficients
C_{c1}	timing(0.89), event(0.75), counter(0.70)
C_{c2}	message(1.00), control(0.86), change(0.55)
C_{c3}	corresponding(0.70), module(0.50), status(0.50)
C_{c4}	validation(0.35), recipe(0.30), unit(0.20)
C_{c5}	information(0.82), counter(0.81), data(0.63)

B. Computer-Supported Analysis (RQ2)

We used KH Coder [8] to cluster defects D_{Ah}. KH Coder is one of the major natural language processing tools that can analyze descriptions written in Japanese. Table XIII shows the coverage of defect type T_h by the clusters from defects D_{Ah}. The clusters C_a, C_b, and C_c correspond to clusters obtained by specifying 15, 10, and 5 as the number of clusters, respectively. Clusters C_{a1}, C_{b1}, and C_{c1} had one-to-one correspondences with defect type T_{h1}. One-to-one correspondence was also observed between defect type T_{h5} and clusters C_{a10} and C_{b8} as well as between defect type T_{h6} and clusters C_{a11} and C_{b9}. Clusters C_{a3}, C_{a9}, C_{a12}, C_{b3}, C_{b7}, C_{b10}, C_{c3}, C_{c4}, and C_{c5} corresponded to two or more defect types. Fourteen defects that were not selected for summarizing and abstracting in RQ1 were clustered.

Tables XIV, XV, and XVI show the three words with the top three largest Jaccard similarity coefficients in each cluster. In the tables, each value between brackets is a Jaccard similarity coefficient. A larger value indicates that the word appears frequently in the defects clustered into the same cluster but not in the defects in the other cluster. The words with the largest Jaccard similarity coefficients among a cluster express the corresponding defect types T_{h1}, T_{h5}, and T_{h6}, which had one-to-one correspondence with clusters. The words "timing," "event," and "counter" in clusters C_{a1}, C_{b1}, and C_{c1} express defect type T_{h1} "Lack of strict timing in the definition for counting the number of objects in production." Clusters C_{a10} and C_{b8} have the same words that express defect type T_{h5}. Also, clusters C_{a12} and C_{b10} have the same words that express defect types T_{h7} and T_{h9}. The difference between defect types T_{h7} and T_{h9} is in whether or not conditions are defined for the manufacturing process of converting.

Defect type T_{h2}, "Lack of consideration of changes from the previous versions" includes different characteristic words in clusters C_{a2}, C_{a3}, C_{a4}, C_{a5}, C_{b2}, C_{b3}, C_{b4}, C_{c2}, C_{c3}, and C_{c4}. Clusters C_{a2}, C_{b2}, and C_{c2} all have the words "message," "control," and "change." The clusters consist of the same six similar defects. However, the analysts focused on broader similarities among C_{a2}, C_{a3}, C_{a4}, and C_{a5}.

VI. Discussion

A. Effectiveness of the Inspection Scenarios (RQ1)

The distributions of the severities of defects that have the potential to be detected by inspection scenarios S_h, S_m, and S_l are all distinct. The percentage of high-severity defects detected using inspection scenarios S_h is larger than that with inspection scenarios S_m, and S_l. Therefore, the answer to research question RQ1-1 is that the inspection scenarios S_h detect more high-severity defects than the number detected by inspection scenarios S_m, and S_l.

Inspection scenarios S_h check the strict definitions of specific capabilities, pre-conditions, and the sequence of manufacturing sub-processes that can potentially mislead programmers into incorrect implementations. Most of inspection scenarios S_m focus on the messages on the system screen, GUI layout, and the status of GUI components in order to avoid unintended operations by system users. Most of inspection scenarios S_l verify the existence of the referred documents for programmers during development and of clear information for system users during operation.

The results show that inspection scenarios S_{h1}, S_{h4}, S_{h6}, and S_{h7} detected only high-severity defects. These inspection scenarios focus on strict definitions of specific manufacturing processes and controls. Overlooking defects in the processes and controls might cause manufacturing product failure due to an incorrect implementation of the manufacturing system. Inspection scenarios S_m focus on ambiguous or omitted messages and available controls on the user interface. The percentage of medium-severity defects detected using inspection scenarios S_m is larger than the percentage of those detected using inspection scenarios S_h and S_l. The number of defects detected using inspection scenarios S_l is small because defects D_{Bl} include various types of defects similar to defects D_{Al}.

In the evaluation, the analyst developed as many inspection scenarios as possible; however, in practice, an analyst can select only high-priority defect types, which leads to more effective inspection scenarios and prioritized defect detection.

The answer to the research question RQ1-2 is that inspection scenarios S_{h3}, S_{h5}, S_{h8}, and S_{h9} can be developed with the VBR checklist while others cannot. The VBR checklist aims to identify the critical capabilities of the software. However, as the number of capabilities becomes larger, identifying and prioritizing these critical capabilities become difficult for an analyst without knowing which defects were detected in previous versions. If the analyst tried to cover all critical capabilities, the identified defect types and the developed inspection scenarios would become too general, and as Brykczynski pointed out, general inspection scenarios do not work [3].

B. Computer-Supported Analysis (RQ2)

The Ward clustering method clustered defects D_{Ah} into subsets of defects. Some of the subsets had a one-to-one correspondence with the defect types identified in RQ1. The results indicate that the defect clusters can help an analyst select similar defects. The results also suggest that the words with larger Jaccard similarity coefficients potentially help analyst identify defect types and develop inspection scenarios. The answer to the research question RQ2 is that computer-supported analysis of defects can help an analyst identify defect types.

Discussion with the analysts clarified that some clusters such as C_{a1}, C_{b1}, and C_{c1} were clear enough to identify defect types and that the defects clustered therein had explicit similarities that distinguished them sufficiently from the other clusters. The discussion also clarified that the similarities among the defects of other clusters depended on the number of clusters specified as a parameter to the clustering algorithm. A clustering algorithm that takes the similarity measure as a threshold might provide more explicit clusters. We believe that employing other clustering algorithms and computer-supported approaches for categorizing defects would produce similar results, and this remains a potential area for future study.

Although 14 defects were not used for identifying the defect types in RQ1 by the analysts, these 14 defects were clustered into subsets by the clustering algorithm. The analysts pointed out that the defect descriptions in the subsets included

similar words and could be categorized as defect types. However, inspection scenarios corresponding to the defect types would be ambiguous or too general to be used to detect defects in a subsequent version.

C. Discussions with Practitioners

The authors engaged in discussion with engineers working on the development of the system. The engineers included the software development leader and analysts X and Y. The outcomes of the discussions are as follows:

- Prioritized inspection is possible with inspection scenario S_h. Inspection scenarios S_h assessed the essential and critical capabilities of the manufacturing system. Also, the inspection scenarios examined whether the documented essential and critical capabilities, including the manufacturing process and the sequence of manufacturing sub-processes, were correct or not. The correctness of the documentation reduces rework in system testing.
- The inspection scenarios developed by the VBR checklist could not specify the essential and critical capabilities with a granularity equivalent to inspection scenarios S_h. If an equivalent granularity is required, the number of inspection scenarios obtained by the VBR must be large because there are many similar essential and critical capabilities such as converting, counting, and control capabilities.
- The defects detected in previous software versions are the defects in the essential and critical capabilities for the system. The defects also led to misunderstandings by the programmers and incorrect implementations that required reworking during the system testing phase.
- In the procedures used to answer RQ1, word frequency was used to identify defect types. If required defect data are available, including defect triggers, orthogonal defect classification [5] can be used to identify defect types.
- The words with larger Jaccard similarity coefficients in some clusters in C_a and C_b induced defect types without having an analyst consider every single defect description in detail. Clustering defects potentially reduces the effort required for identifying defect types and developing corresponding inspection scenarios. The words can mitigate barriers for developing inspection scenarios in practice. In fact, when the list of the three words with the largest Jaccard similarity coefficients shown in Tables XIV, XV, and XVI in each cluster was shown on the screen, engineers said "Those are in function X." and "He is good at detecting them."
- In the defect repository, defects D_A and D_B have similar trends, because the changed and added requirements are similar in both versions. Our procedure depends on similarities existing between previous versions of the software and the subsequent version.

D. Threats to Validity

The identified defect types and the developed inspection scenarios might be too dependent on the particular analyst performing the task; however, in the evaluation, we used frequently used words to abstract and summarize defects in the evaluation to reduce the dependency on the analyst. Moreover, we asked another analyst to ensure that the defect types and inspection scenarios were adequate.

Because the defect repository in the evaluation included two releases without any large changes in the inspection process and the development process, the defects D_A and the defects D_B were considered to have similar types of defects. Large process changes or process improvements between previous software versions and a subsequent version must be considered when using the defect repository of defects detected in previous software versions.

The defects D_{Ah} and D_{Bh} included defects specific to the manufacturing process and controls. That could be one of the causes for the larger percentage of high-severity defects detected using inspection scenarios S_h. Based on our previous research of defect repositories, including the articles [12] and [13], we believe that this tendency is common among defect repositories. Further investigation and discussion of similar tendencies in other defect repositories is required to generalize the results of our study.

VII. Conclusion

This paper evaluated the effectiveness of inspection scenarios developed by abstracting and summarizing defects detected during software inspections of previous software versions. First, an analyst with knowledge and experience of the target software identifies defect types from high-severity defects detected in previous software versions. Then, the analyst develops an inspection scenario for each defect type. In software inspections of a subsequent version, inspectors try to detect defects using the developed inspection scenarios. The inspection scenarios enable inspectors to conduct prioritized defect detection.

The evaluation investigated 456 defects detected in inspections of two versions of an industrial manufacturing system. Each defect had one of the three severities (high, medium, or low). The defects were collected from the earlier version (defects D_A), which included 59 high-severity defects, as well as from the later version (defects D_B), which included 48 high-severity defects. Analyst X identified nine defect types out of the high-severity defects in D_A and developed the corresponding inspection scenarios. To reduce the dependency on analyst X, analyst Y ensured that the identified defect types and the developed inspection scenarios were adequate.

Analyst X judged whether each inspection scenario could potentially detect defects in D_B. The inspection scenarios from high-severity defects were found to potentially detect 118 defects, including 46 high-severity defects (39.0%). The inspection scenarios from medium-severity defects could potentially detect 102 defects, including six high-severity defects (5.9%). The inspection scenarios from low-severity defects could no high-severity defects. Also, analyst Y examined whether each inspection scenario could be obtained from the checklist proposed by the VBR. Analyst Y determined that the five of the nine inspection scenarios developed from high-severity defects could not be obtained with the VBR checklist.

To evaluate the feasibility of computer-supported inspection scenario development, defects were categorized by a clus-

tering algorithm and compared to the identified defect types by the analysts. The results showed that some clustered defects had a one-to-one correspondence with identified defect types and that other clustered defects were subsets of identified defect types. The results indicate that cluster analysis could help analysts develop inspection scenarios and could reduce the effort required to identify defect types.

REFERENCES

[1] Z. Abdelrabi, E. Cantone, M. Ciolkowski, D. Romach, Comparing Code Reading Techniques Applied to Object-Oriented Software Frameworks with Regard to Effectiveness and Defect Detection Rate, Proceedings of the 2004 International Symposium on Empirical Software Engineering, pp.239-248(2004)

[2] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sivert Sorumgard, M. V. Zelkowitz, The Empirical Investigation of Perspective-Based Reading, International Journal of Empirical Software Engineering, vol. 1,no. 2, pp. 133-164(1996)

[3] B. Brykczynski, A Survey of Software Inspection Checklists, ACM SIGSOFT Software Engineering Notes vol. 24, no. 1, pp. 82-89(1999)

[4] Y. Chernak, A Statistical Approach to the Inspection Checklist Formal Synthesis and Improvement, IEEE Transactions on Software Engineering, vol.22, no.12, pp.866-874(1996)

[5] R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, D.S. Moebus, B.K. Ray, and M. Wong, Orthogonal Defect Classification - A Concept for In-Process Measurements. IEEE Transactions on Software Engineering, vol. 18, no. 11, pp. 943-956(1992)

[6] C. Denger, M. Ciolkowski, F. Lanubile, Investigating the Active Guidance Factor in Reading Techniques for Defect Detection, Proceedings of the 2004 International Symposium on Empirical Software Engineering, pp. 219-29(2004)

[7] M.E. Fagan, Design and Code Inspection to Reduce Errors in Program Development, IBM System Journal, vol.15, no.3, pp.182-211(1976)

[8] K. Higuchi, KH Coder, http://khc.sourceforge.net/en/

[9] P. Jaccard, The Distribution of the Flora in the Alpine Zone, New Phytologist, vol.11, no. 2, pp. 37-50(1912)

[10] F. Lanubile, F. Shull, V. R. Basili, Experimenting with Error Abstraction in Requirements Documents, In Proceedings of the 5th International Symposium on Software Metrics, pp. 114-121(1998)

[11] K.Lee, B. Boehm, Empirical Results from an Experiment on Value-based Review (VBR) Processes, Proceedings of International Symposium on Empirical Software Engineering 2005 pp. 17-18(2005)

[12] T. Matumura, S. Morisaki, A. Monden, K. Matsumoto, Analyzing Factors of Defect Correction Effort in a Multi-vendor Information System Development, Journal of Computer Information Systems, vol.49, No.1, pp.73-80(2008)

[13] S. Morisaki, A. Monden, T. Matsumura, H. Tamada, K. Matsumoto, Defect Data Analysis Based on Extended Association Rule Mining, Poceedings of the Fourth International Workshop on Mining Software Repositories, pp.1-8(3) (2007)

[14] S. Morisaki, Y. Kamei, K. Matsumoto, An Experimental Evaluation of the Effectiveness of Specifying A Defect Type in Software Inspection, Journal of Information and Media Technologies, vol.6, no.4, pp.173-178(2011)

[15] A. Porter, L. Votta and V.R. Basili, Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment, IEEE Transactions on Software Engineering, vol.21, no.6, pp. 563-575(1995)

[16] E. Shihab, A. Mockus, Y. Kamei, B. Adams, A. E. Hassan, High-impact Defects: A Study of Breakage and Surprise Defects. In Proceedings of the 19th Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 300-310(2011)

[17] T. Thelin, P. Runeson, B. Regnel, Usage-based Reading: An Experiment to Guide Reviewers with Use Cases, Information and Software Technology, vol. 43, no. 15, pp.925-938(2001)

[18] J. H. Ward, Hierarchical Grouping to Optimize an Objective Function, Journal of the American statistical association vol.58, no.301, pp. 236-244(1963)

[19] G. S. Walia, J.C. Carver, Using Error Abstraction and Classification to Improve Requirement Quality: Conclusions from a Family of Four Empirical Studies, Journal of Empirical Software Engineering, vol.18, no.4, pp.625-658(2013)