Handover Anti-patterns

Kei Ito[1], Hironori Washizaki[2], Yoshiaki Fukazawa[3]

[1] Waseda University

[2] Global Software Engineering Laboratory, Waseda University

[3] Global Software Engineering Laboratory, Waseda University

k-win@toki.waseda.jp, washizaki@waseda.jp, fukazawa@waseda.jp

Abstract. *Every organization undergoes personnel changes that induce handover activities. Most business people are familiar with the concept of a handover. Issues with handovers became apparent in Japan in 2007 as many people from the Baby Boomer Generation retired simultaneously. Although effective handovers are crucial for seamless business operations during personnel changes, the preferable elements for an ideal handover are ambiguous and little research has been conducted. Our research focuses on anti-patterns, which identify the causes of an unsuccessful handover. Paradoxically, the handover anti-pattern allows preferable elements for handover to become clear. Herein we introduce three anti-patterns, which were elucidated from a workshop to collect information about unsuccessful handovers.*

Categories and Subject Descriptors

•Social and professional topics → Project and people management

•Software and its engineering → Maintaining software

• Software and its engineering → Software verification and validation

General Terms

Anti-patterns, Design patterns, Project management

Keywords

Handover, Anti-patterns, Pattern Language, People Management, Maintaining software

1. Introduction

Handover is a process of transferring responsibilities from the predecessor to the successor [AM10a] [AM10b]. Most business personnel are familiar with this concept. Despite its importance, little is known about handover problems and few publications deal with this handover process. One study [AM10b], which investigated core problems of handovers from a developer to the maintainer, mentions that insufficient knowledge is the main handover problem. Moreover, information sharing is a complex problem. Some research has investigated the complexity of information [TE08], but none has focused on handovers problems due to complexities with information sharing.

Our research focuses on information-sharing problems during a handover. Because handovers are common, this research may be applicable to numerous fields, and those without a systems background may also enjoy this paper. This paper aims to identify concrete problems with handovers. One way to do this is to define anti-patterns. The term anti-pattern is from Design Patterns. Although design patterns highlight desirable solutions, which are considered highly reliable and effective, anti-patterns highlight negative solutions. Anti-patterns provide the necessary knowledge to prevent or recover from undesired situations. Examples of anti-patterns include death march, god class, and vendor lock-in[H1998]. That is, handover anti-patterns are maps of dangerous scenarios and can help detect handover problems. Herein we propose an approach to elucidate the elements necessary for a preferable handover by defining common handover anti-patterns. Using these maps, project managers can prevent undesirable situations due to unsuccessful handovers and construct reliable organization policies that are unaffected by personnel changes. Finally, we strive to develop a pattern language for handovers solve the problem identified in the anti-patterns. Consequently, the anti-patterns in this paper are the “pattern seeds” for a pattern language.

The rest of the paper is organized as follows. Section 2 introduces our anti-pattern template. Each anti-pattern contains five items: name, scenario, main cause, refactored solution, and refactored scenario. Section 3 discusses handover using activity diagrams, class diagrams, and object diagrams. Section 4 extracts three common anti-patterns. Finally, Section 5 concludes this paper and states possible future works.

2. Handover Anti-pattern Template

Our anti-pattern has two solutions. The first solution provides an anti-pattern problem, a solution commonly used by organizations but which is ineffective. The second one provides a refactored solution, a strategy to help improving anti-pattern situation[WRHT98]. Here are the five items each handover anti-pattern contains.

- Name

  Concise expression of the anti-pattern situation contained in the class diagram

- Scenario

  An anti-pattern scenario based on a case study expressed in the object diagram

- Main causes

  Description of the anti-pattern causes

- Strategies to prevent and recover from the anti-pattern

To prevent the anti-pattern from happening, the predecessor should apply Strategy A before leaving the post. In case he fails to do that, his successor should apply Strategy B to recover from the anti-pattern. Strategy A is a measure to prevent the anti-pattern situation while Strategy B aims to correct the problem posed by the anti-pattern.

Table 1 Strategies to prevent and recover from the anti-pattern

	Measure to prevent the anti-pattern situation	Measure to recover from anti-pattern situation
Predecessor	Strategy A	-
Successor	-	Strategy B

- Refactoring scenario

This item is based on the anti-pattern scenario and the prevention and recovering strategies. The object diagram describes the effectiveness of a refactored solution

3. Handover Model

Handovers involve an information-sharing process. Here we explain this process with model and an example to illustrate handover activities and necessary elements. In addition, we propose a model to define the handover elements.

- Example

Staff A has been working on system X. Staff A, who is leaving this post, is being replaced by Staff B. The handover elements involve:

- Predecessor: Staff A
- Successor: Staff B
- Handover target: Operation of business management system X

Figures 1-3, show our proposed activity diagram, class diagram and object diagram, respectively.

3.1 Handover Activity Diagram

The handover activity diagram is used to define the handover activities.

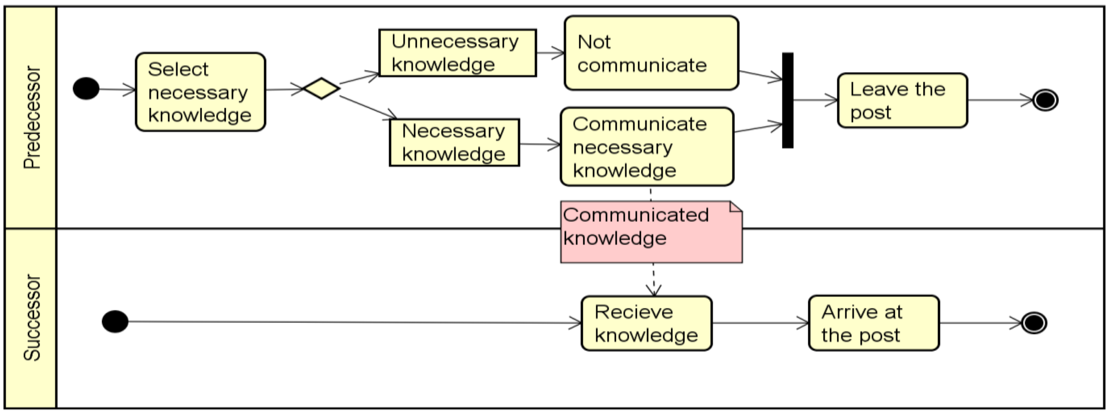

Figure 1 Handover activity Diagram

The handover activities are divided into following tasks:

(1) Staff A (Predecessor) selects the necessary knowledge and unnecessary knowledge.

(2) Staff A communicates necessary knowledge to Staff B (Successor).

(3) Staff B receives the necessary knowledge from Staff A.

(4) Staff A is replaced by Staff B.

This diagram contains six activities and two actors (Predecessor and Successor) and three activities (Select necessary knowledge, Communicate necessary knowledge and receive knowledge).

3.2 Handover Class Diagram

Next we propose a handover class diagram to describe these activities and elements in more detail.

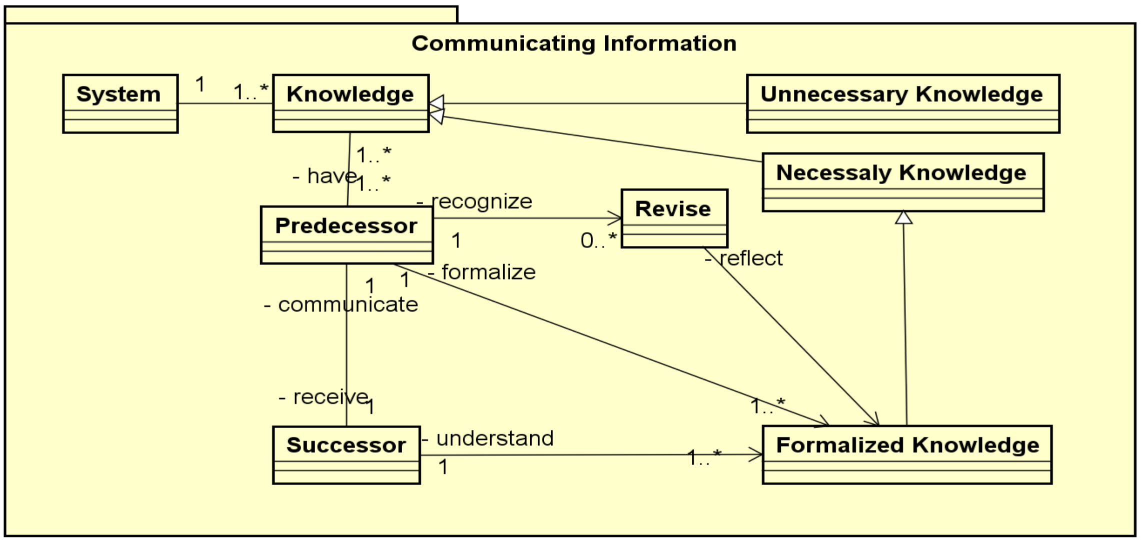

Figure 2 Handover class diagram

There are eight classes in the handover diagram:

(1) System Class:
- Target of the Knowledge Class

(2) Knowledge Class:
- Knowledge of the System Class owned by the Predecessor Class
- Two sub-classes: Unnecessary Knowledge Class and Necessary Knowledge Class

(3) Unnecessary Knowledge Class:
- Unnecessary Knowledge for the system.
- Sub-class of the Knowledge Class

(4) Necessary Knowledge Class:
- Necessary Knowledge for the system.
- Sub-class of the Knowledge Class

(5) Formalized Knowledge Class:
- Formalized shape of necessary knowledge.
- Sub-class of the Necessary Knowledge Class.

(6) Revise Class:
- Recognized by the Predecessor Class and reflected in the Formalized Knowledge Class.

(7) Predecessor Class:
- Communicates necessary knowledge to the Successor Class

(8) Successor Class:
- Receives necessary knowledge from the Predecessor Class.

3.3 Handover Object Diagram

We propose the object diagram to complement the class diagram.

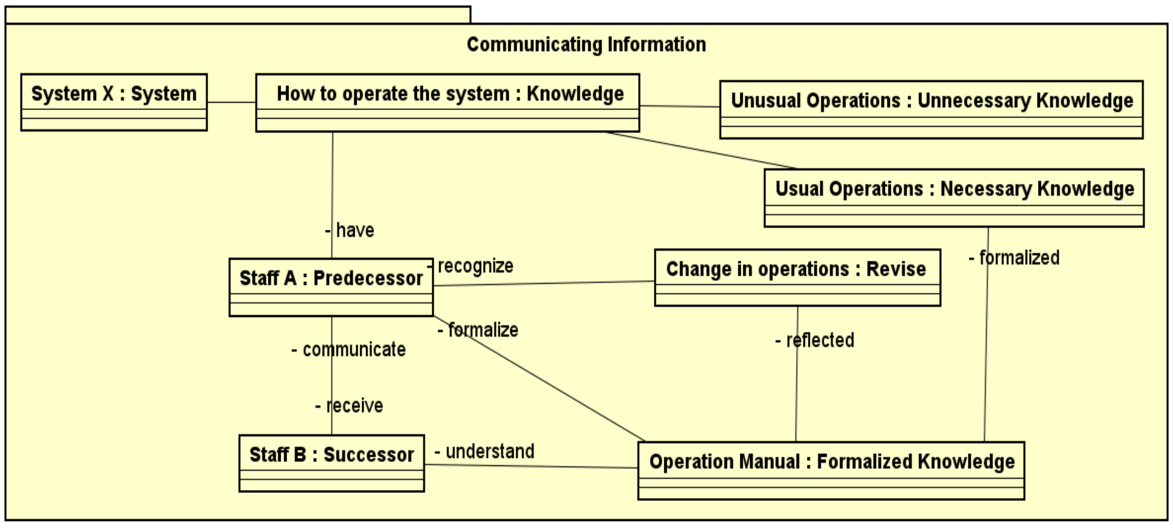
Communicating Information
System X : System
How to operate the system : Knowledge
Unusual Operations : Unnecessary Knowledge
Usual Operations : Necessary Knowledge
- have
Staff A : Predecessor
- recognize
Change in operations : Revise
- formalized
- formalize
- communicate
- reflected
- receive
Staff B : Successor
- understand
Operation Manual : Formalized Knowledge

Figure 3 Handover object diagram

The handover object diagram has the following eight classes:

(1) System X: System Class
(2) How to operate the system: Knowledge Class
(3) Unusual operations: Unnecessary Knowledge Class
(4) Usual operations: Necessary Knowledge Class
(5) Operation manual: Formalized Knowledge Class
(6) Change in operations: Revise Class
(7) Staff A: Predecessor Class
(8) Staff B: Successor Class

4. Handover Anti-pattern

The handover anti-pattern is classified into two classes. Handover activity consists of two activities, select necessary knowledge, and communicate necessary knowledge. The failures of these activities bring handover anti-pattern. In this section, we introduce three anti-patterns, *Unsupported to review*, *Background knowledge is unclear and Necessary knowledge is omitted*. These are caused by failure of select necessary knowledge activity.

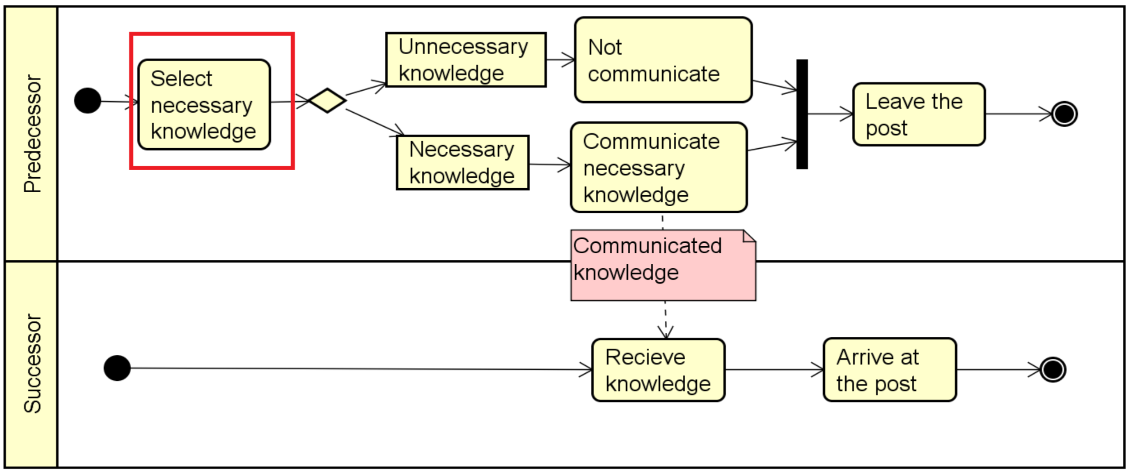

Figure 4 Handover activity map

Failure of the select necessary knowledge activity results in defective knowledge being transmitted. Consequently, predecessor communicates defective knowledge to successor. Defective knowledge has two elements; incorrectness and insufficiency. We explain relationship between these elements and each anti-pattern by map of select necessary knowledge anti-pattern described by class diagram.

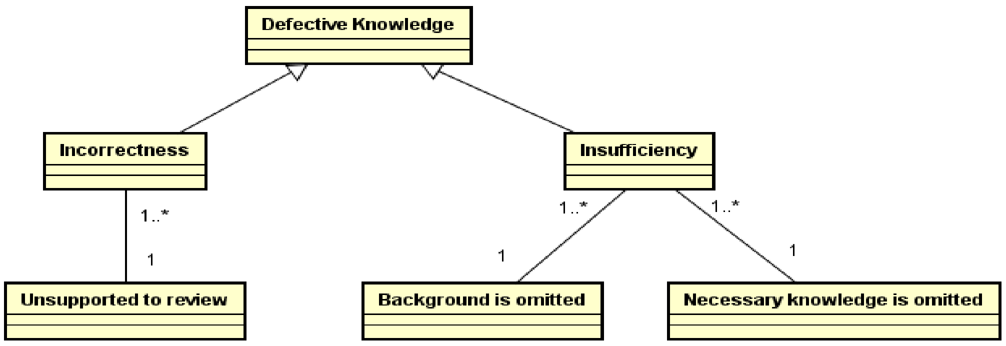

Figure 5 Map of select necessary knowledge anti-pattern

Incorrectness has a relationship between Unsupported to review. Insufficiency has a relationship between Background is omitted and Necessary knowledge is omitted. Next, we explain each anti-pattern in detail from the next section.

4.1 Unsupported To Review

A review conference, which denotes defects that must be revised, is an opportunity to correct defects in a document. However, documents are not always revised after a review conference. Sharing unrevised documents during the handover process tends to cause issues when the successor assumes responsibility for a task.

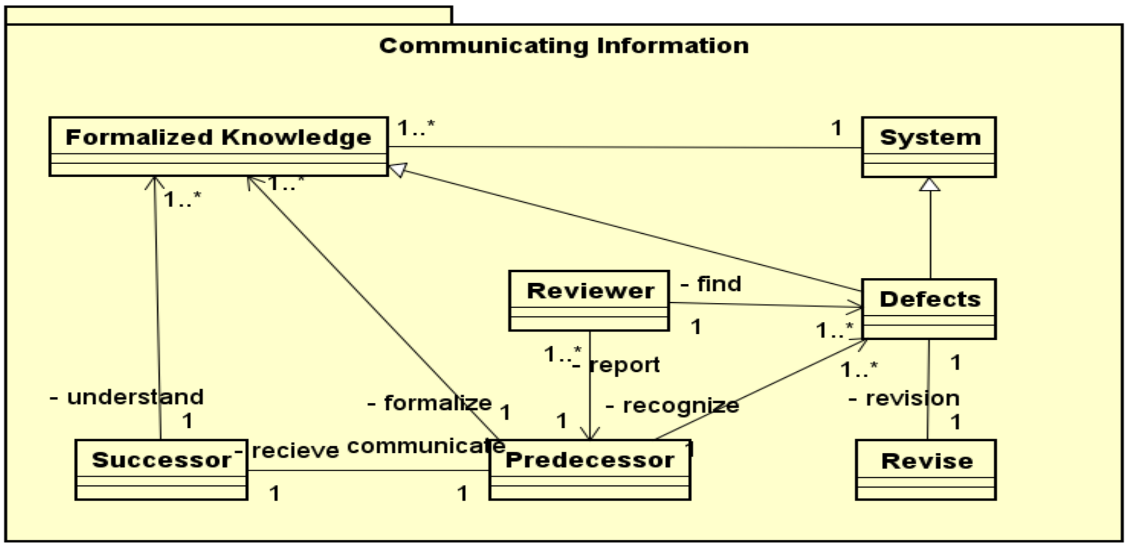

Figure 6 “Unsupported to review” class diagram

- Scenario
  (1) Staff A is in charge of operations of System X.
  (2) During a personnel change, Staff B assumes this task from Staff A.
  (3) Staff A makes operation manual of System X.
  (4) Staff C reviews the document and find incorrect operation.

(5) Staff C reports the defects to Staff A.
(6) Staff A recognizes the defects, but does not revise the document.
(7) Staff B operates System X using a defective manual and fails to operate System X appropriately.

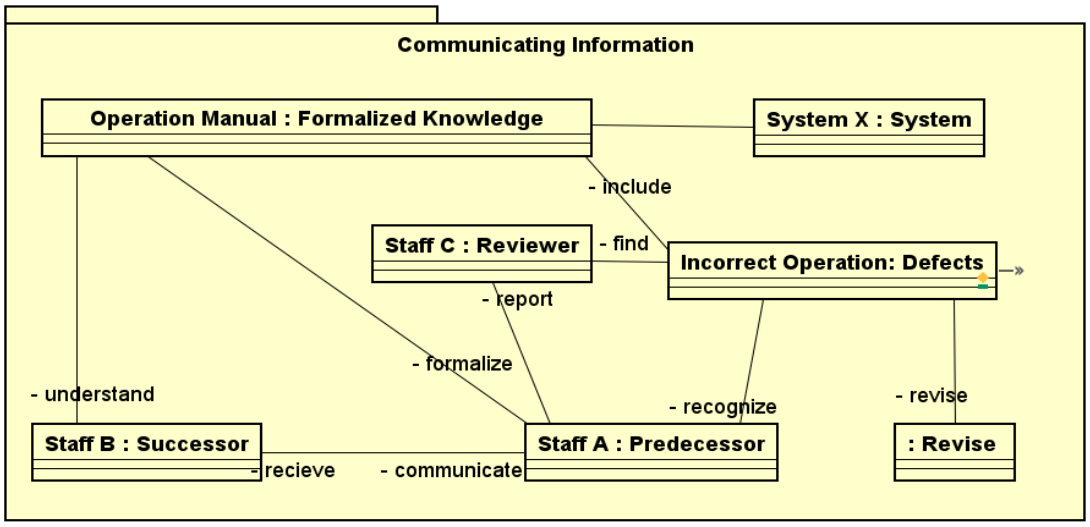

Figure 7 “Unsupported to review” object diagram

- Main cause

This anti-pattern occurs because the predecessor does not revise the documents. Often the predecessor misunderstands the document status and cannot determine whether a document is updated because a method to verify the document status does not exist. Thus, defective document is shared during a handover, preventing the successor from appropriately executing the post.

- Strategies to prevent and recover from the anti-pattern

Because the main cause is the lack of a method to check the documents, introducing an update history of the documents is an effective solution.

Table 2 Strategies to prevent and recover from the anti-pattern of “unsupported to review”

	Measure to prevent the anti-pattern situation	Measure to recover from anti-pattern situation
Predecessor	･ Record the update history in the document. ･ Check the status frequently, and update the documents as necessary.	-
Successor	･ Check the document status before the predecessor leaves, and ask the predecessor if the documents are updated	-

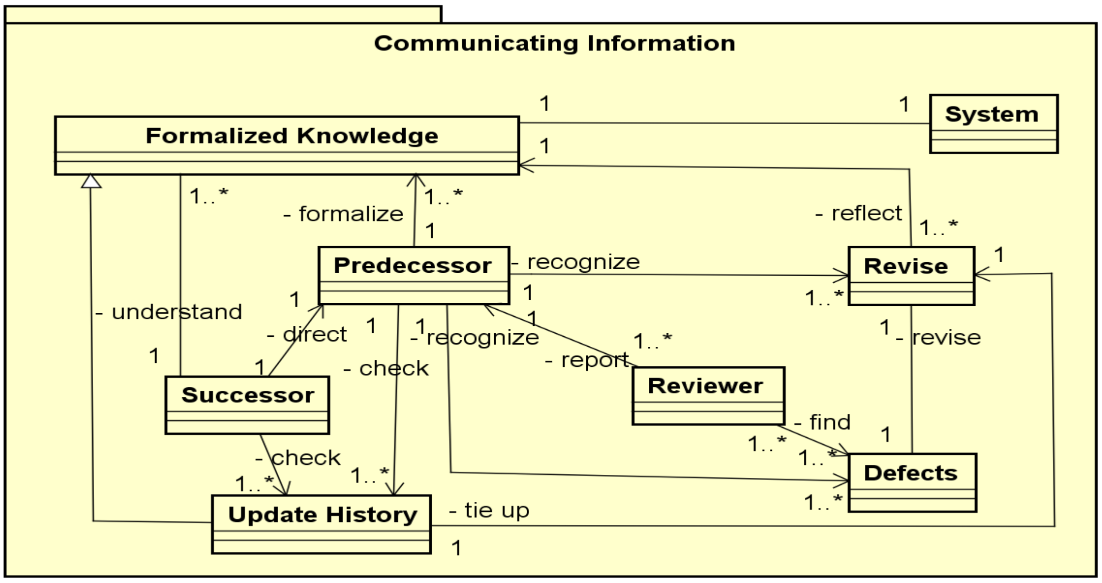

Figure 8 Refactored “unsupported to review” class diagram

- Refactored scenario
  (1) Staff A is in charge of operations of System X.
  (2) During a personnel change, Staff B assumes this task from Staff A.
  (3) Staff A makes operation manual of System X.
  (4) Staff C reviews the document and find incorrect operation.
  (5) Staff C reports the defects to Staff A.
  (6) Staff A recognizes the defects, but does not revise the document.
  (7) Staff B receives the manual and notes that the document is not updated.
  (8) Staff A update the documents to remove defects and updates the history.

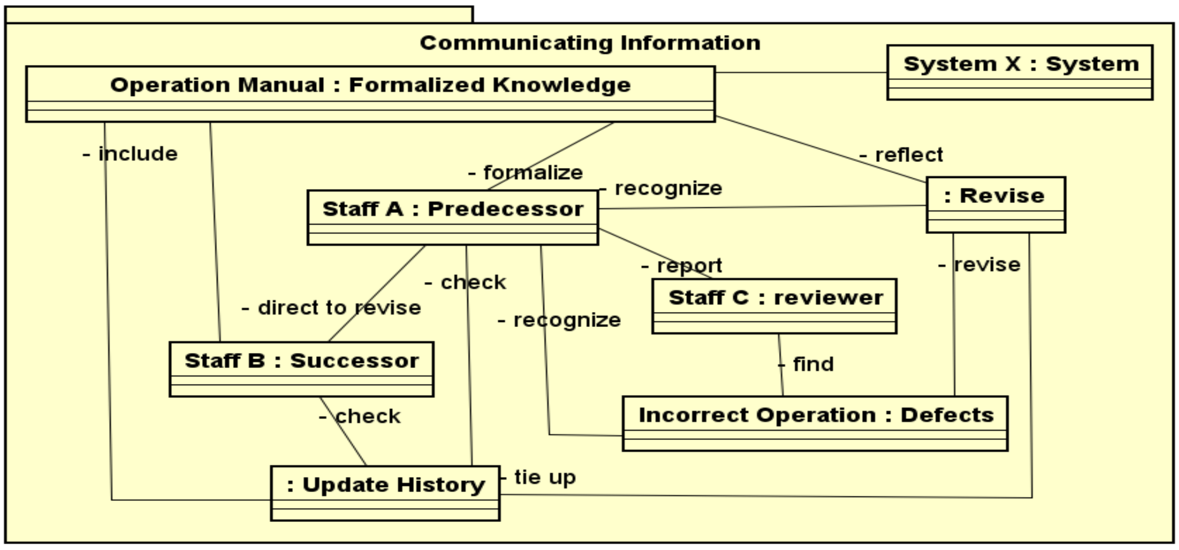

Figure 9 Refactored “unsupported to review” object diagram

4.2 Background Knowledge Is Unclear

All systems have background knowledge such as design concepts, requests from customers, and restrictions regarding budgets or technical levels. Although background knowledge indirectly affects the system, background knowledge tends to be lost because it is not recorded in the handover document.

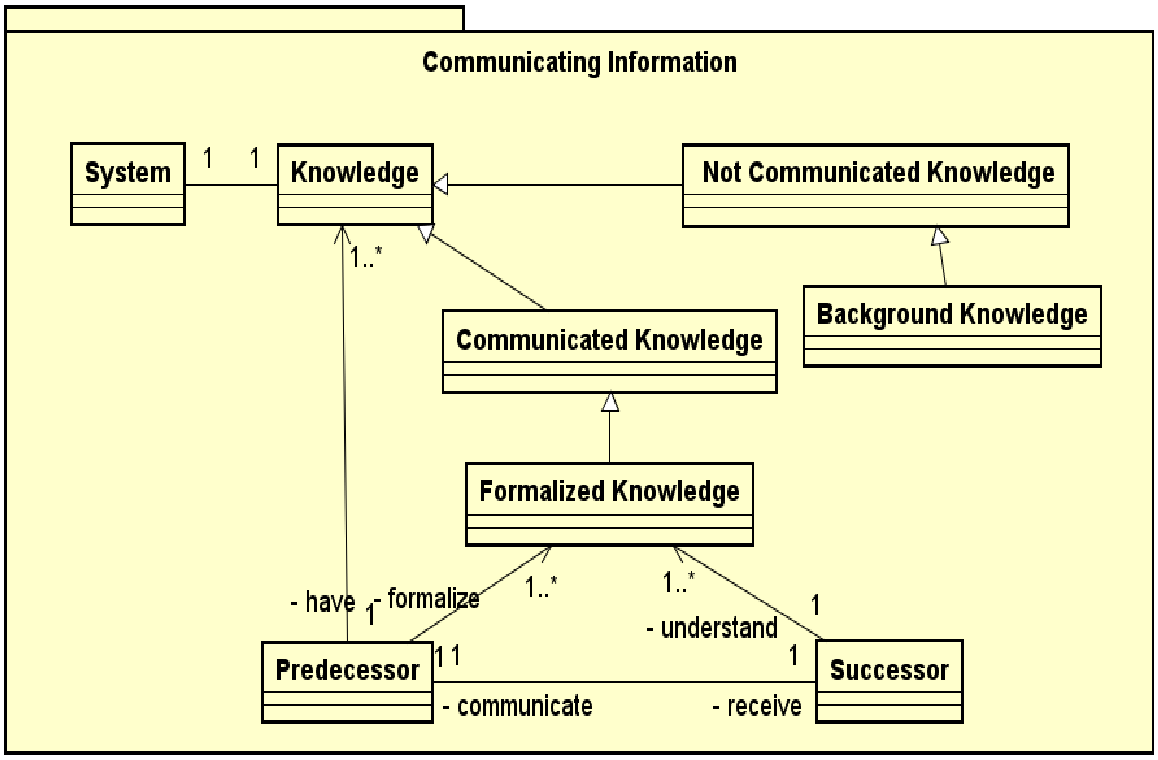

Figure 10 “Background knowledge is unclear” class diagram

- Scenario
  (1) System X is developed and operated for long periods by Staff A.
  (2) During a personnel change, Staff B assumes this task from Staff A.
  (3) After completing the handover, Staff B assumes the post from Staff A,
  (4) A short time later, Staff B finds an ambiguous output value.
  (5) The reason for the ambiguity is not present in any of the documents and Staff A is no longer at the company.
  (6) The background knowledge and know-how to deal with the ambiguous output value are lost.

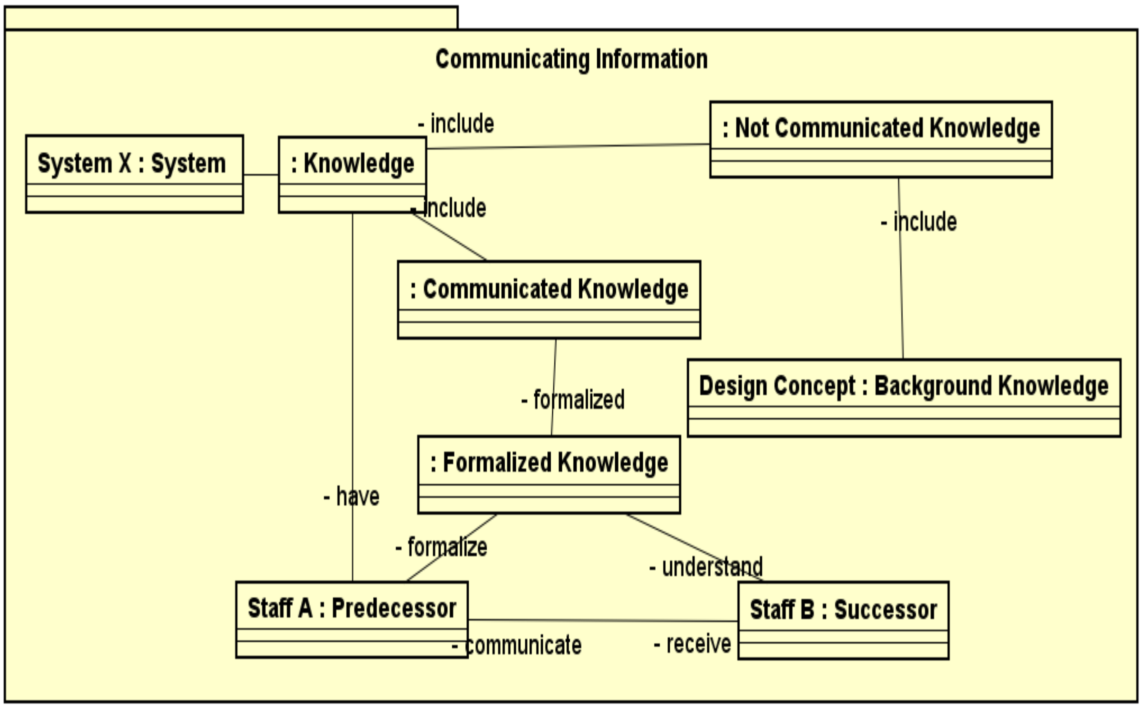

Figure 11 “Background knowledge is unclear” object diagram

- Main causes

  Background knowledge affects the system, but it tends to be excluded during a handover, and over time, fewer people understand the background knowledge. Moreover, background knowledge is not recorded in the specification documents.

- Strategies to prevent and recover from the anti-pattern

Background knowledge is not recorded in the specifications. However, the background knowledge is often recorded in other data, such as review of development or the permission of the system because these types of data pertaining to the purpose and suitability of the system.

Table 3 Strategies to prevent and recover from the anti-pattern of “background is unclear”

	Measure to prevent the anti-pattern situation	Measure to recover from anti-pattern situation
Predecessor	· Preserve the data of the review and the permission.	-
Successor	· Verify that the data of review and the permission exists before the predecessor leaves.	· Trace the data of the review and the permission data, etc.

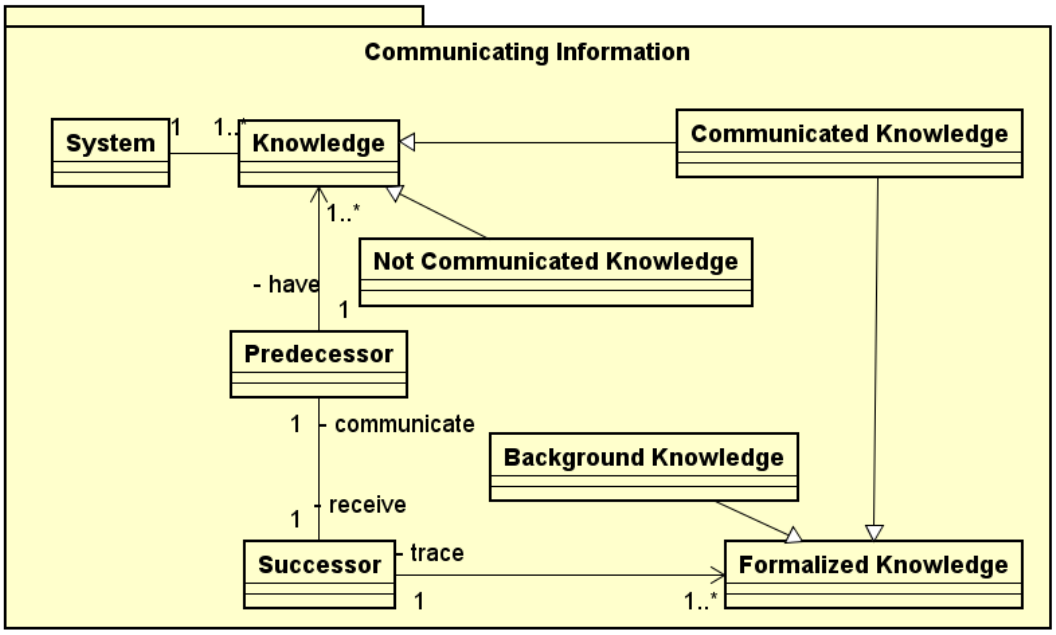

Figure 12 Refactored "background knowledge is unclear" class diagram

- Refactored scenario

(1) System X is developed and operated for long periods by Staff A.

(2) During a personnel change, Staff B assumes this task from Staff A.

(3) After completing the handover, Staff B assumes the post from Staff A,

(4) A short time later, Staff B finds an ambiguous output value.

(5) The reason for the ambiguity is not present in any of the documents and Staff A is no longer at the company.

(6) Staff B traces the review data or other data to find the background knowledge of the ambiguous output value.

(7) Staff B is able to deal with the output value appropriately.

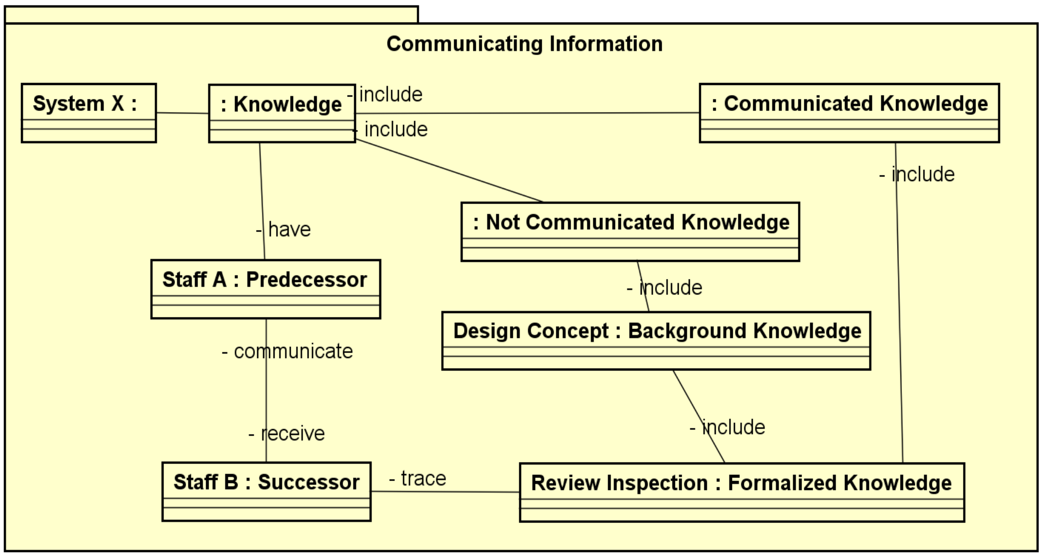

Figure 13 Refactored “background knowledge is unclear” object diagram

4.3 Necessary Knowledge Is Omitted

The predecessor chooses the necessary knowledge to be shared, and records this information in documents. However, necessary knowledge may be omitted from the handover documents. In this case, omitted necessary knowledge is not passed to the successor, and prevents the successor from appropriately executing the post.

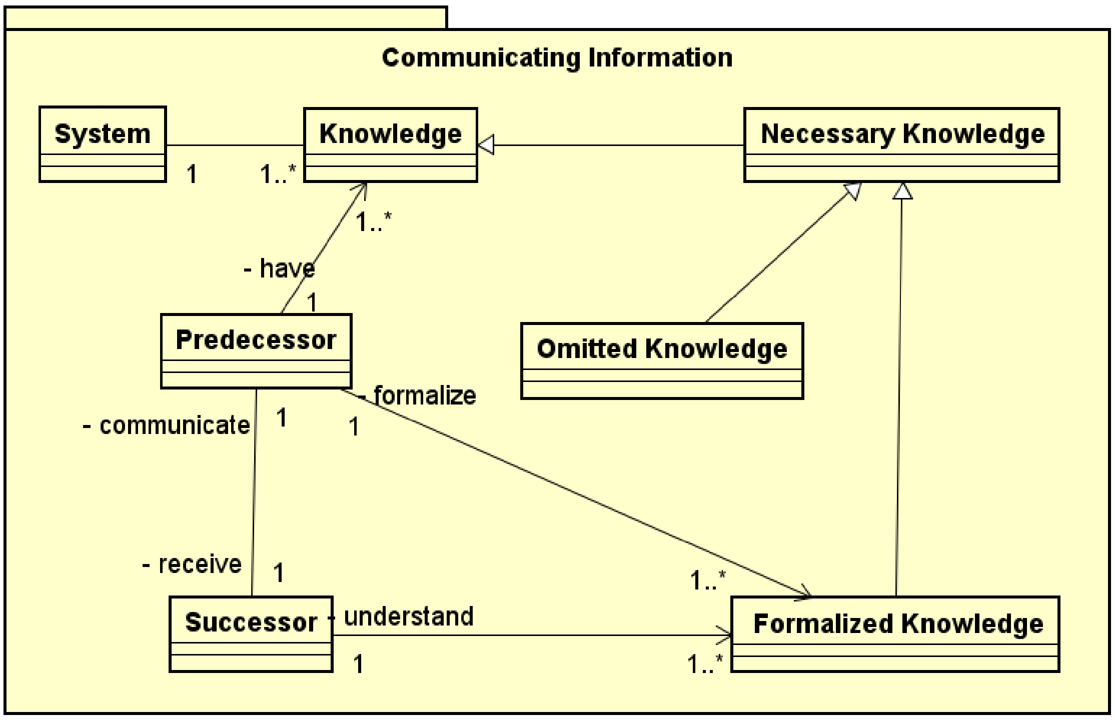

Figure 14 “Necessary knowledge is omitted” class diagram

- Scenario
  (1) Staff A maintains System X.
  (2) During a personnel change, Staff B assumes this task from Staff A.
  (3) Staff A handovers to Staff B for his work, but part of the necessary procedure to back up the system data (operation Y) is omitted from the handover.
  (4) After completing the handover, Staff B assumes the post from Staff A,
  (5) A short time later, the backup function of system X gets weaker suddenly and the necessary information is unknown.

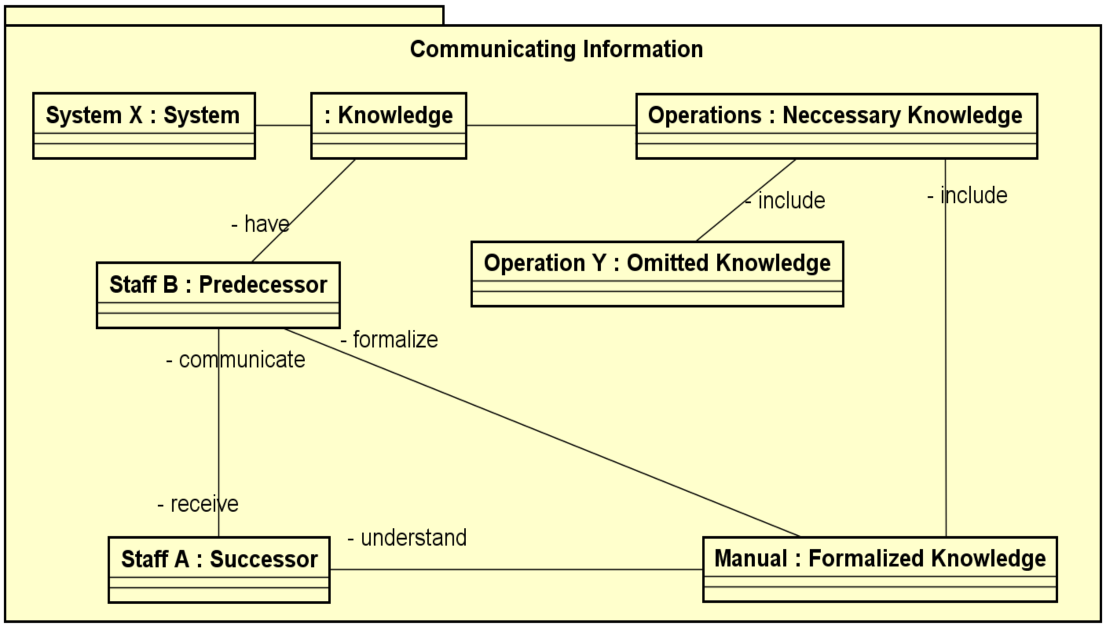

Figure 15 “Necessary knowledge is omitted” object diagram

- Main cause

  The absence of a method to verify omitted knowledge objectively is the main cause of this anti-pattern. Although the predecessor verifies whether the handover documents contain all necessary knowledge, a self-check tends to be subjective and ad-hoc review, making it difficult for the predecessor to determine if knowledge is omitted.

- Strategies to prevent and recover from the anti-pattern

  To improve the review, there are various software reading techniques developed such as Checklist-Based Reading [HTY15]. The table below shows effective measures to verify that a handover includes all necessary information.

Table 4 Strategies to prevent and recover from the anti-pattern of “necessary knowledge is omitted”

	Measure to prevent the anti-pattern situation	Measure to recover from anti-pattern situation
Predecessor	・ Create a checklist for handover items. ・ Check the list by predecessor himself and a third party. If knowledge is omitted, revise the documents.	-
Successor	・ Check the list and if knowledge is omitted, confirm with the predecessor.	-

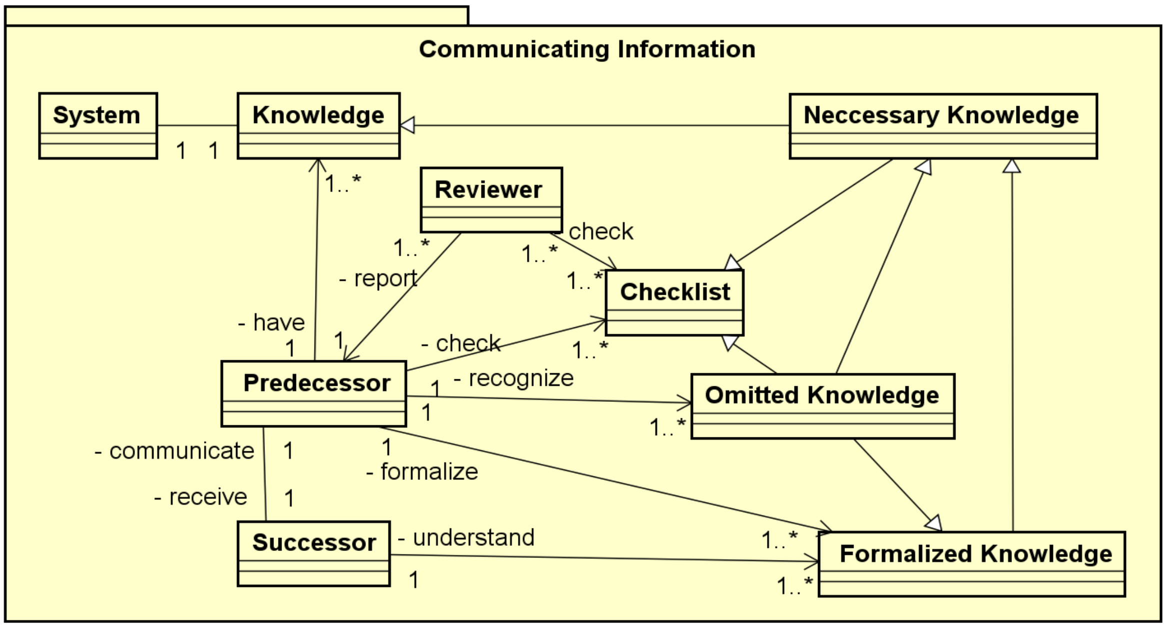

Figure 16 Refactored "necessary knowledge is omitted" class diagram

- Refactored scenario

(1) Staff A maintains System X.

(2) During a personnel change, Staff B assumes this task from Staff A.

(3) Staff A creates a checklist for the handover and then verifies the documents

(4) Staff A finds that the necessary procedure for the backup is omitted and revises the documents.

(5) Staff A handovers to Staff B for his work.

(6) After completing the handover, Staff B assumes the post from Staff A,

(7) Staff B maintains System X appropriately.

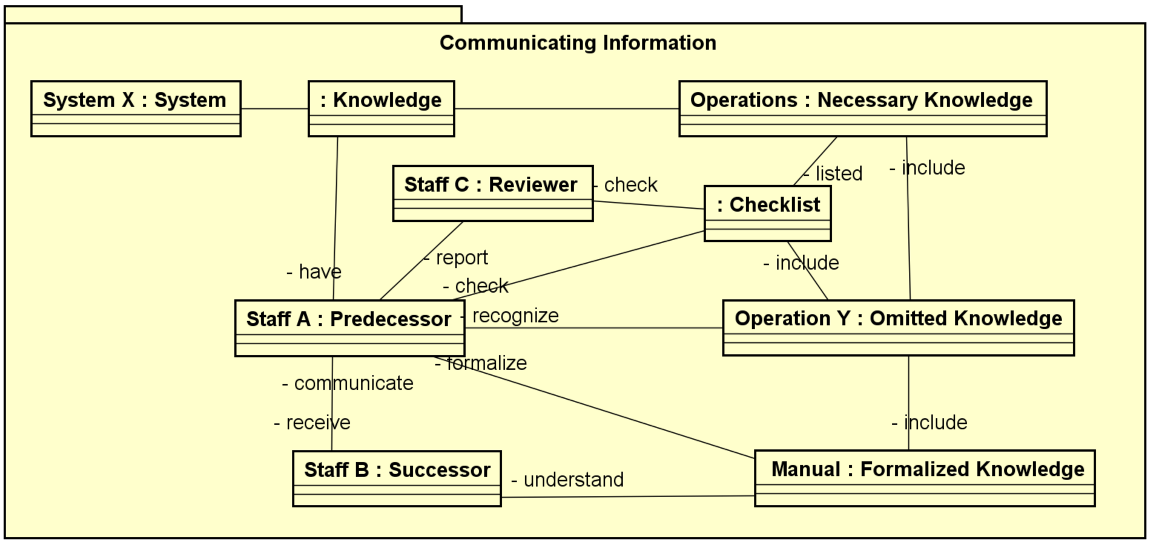

Figure 17 Refactored "necessary knowledge is omitted" object diagram

Conclusion And Future Work

Herein we propose three handover anti-patterns to describe concrete problems in handovers as well as three models to explain handover anti-pattern situations. These models help define handover problems. Each anti-pattern contains refactored solutions to prevent or recover from the anti-pattern situation. The proposed refactored solutions seem effective. The main purpose of the pattern is to construct the relations between each pattern [JN05] [ASMMIS77]. As a next step, we investigate each anti-pattern in more detail and determine their relations and develop a handover pattern language to solve the problems described in this paper.

However, three problems remain. First, we use UML to explain the handover anti-pattern. UML is useful for people with a systems background, but it is difficult for others to understand. Thus, we need to derive more concrete scenarios to describe the pattern language for handovers. Second, this paper uses simple linear models, but activities are more complex and repetitive. The models in this paper do not sufficiently express the concrete activities and repetition in handovers. Consequently, the models need to be improved. Finally, the effectiveness of the proposed solutions has not been verified. We plan to evaluate the effectiveness via questionnaires and use the results to propose an improved language for handovers.

Acknowledgement

We first thank all the participants of the workshop to collect information about unsuccessful handover. We would like to thank Joseph W. Yoder for helping us to improve our paper and future work. Also, we appreciate towards our Foutse Khom, who gave us helpful comments through shepherding process. And lastly, we thank Asian PLoPers who gave us the wonderful comments during the workshop.

References

[AM10a] Ahmad Salman Khan., Mira Kajki-Mattsson. 2010. “Taxonomy of Handover Activities,” 11th International Conference on Product Focused Software 2010 (PROFES’10), Limerick, Ireland, 2010., 131-134

[AM10b] Ahmad Salman Khan., Mira Kajki-Mattsson. 2010. “Core handover Problems,” 11th International Conference on Product Focused Software 2010 (PROFES’10), Limerick, Ireland, 2010., 135-139

[TE08] T. Grandon Gill., Eli Cohen. 2008. “Research themes in complex Informing”, Informing Science: the International Journal of an Emerging Transdiscipline. 11, 2008, 147-164

[H1998] Hays W. McCormick., 1998. *AntiPatterns, a Brief Tutorial*, http://www.antipatterns.com/briefing/index.htm

[WRHT98] William J. Brown., Raphael C. Malveau., Hays W. “Skip” McCormick Ⅲ., Thomas J. Mowbray. 1998. Anti-Patterns. John Wiley & Sons

[HTY15] Hironori Washizaki., Tian Xia., Yoshiaki Fukazawa. 2015. “Introducing Software Reading Techniques into Pattern Writer’s Workshop: Checklists and Perspectives,” 4th Asian Conference on Pattern Languages of Programs2015 (Asian PLoP2015), Tokyo, Japan, 2015

[JN05] James O. Coplien., Neil B. Harrison. 2005. Organizational Patterns of Agile Software Development. Pearson Prentice Hall

[ASMMIS77] Alexander, Christopher., Sara Ishikawa., Murray Silverstein., with Max Jacobson., Ingrid Fiksdahl-King., Shlomo Angel. 1977. A Pattern Language: Towns, Buildings, Construction. New York: Oxford University Press