Supporting Q&A Processes in Requirements Elicitation: Bad Smell Detection and Version Control

Yui Imahori[1], Junzo Kato[2], Shinpei Hayashi[3], Atsushi Ohnishi[4], and Motoshi Saeki[1]

[1] Nanzan University yui.imahori@apps.nise.org, saeki@nanzan-u.ac.jp
[2] Independent Consultant junzo.katou@gmail.com
[3] Tokyo Institute of Technology hayashi@c.titech.ac.jp
[4] Ritsumeikan University ohnishi@is.ritsumei.ac.jp

Abstract. In the process of developing requirements specifications, a requirements analyst conducts question-and-answer (Q&A) sessions iteratively to incrementally make more complete initial requirements preobtained from stakeholders. However, iterated Q&A sessions often have some problems leading to a final requirements specification of lower quality. This paper presents the usage of a graph database system to identify *bad smells* in Q&A processes, which are symptoms leading to a lower quality product, and to control the versions of a list of requirements through the activities. In this system, the records of the Q&A activities and the requirements lists are structured and stored in a graph database Neo4j. Cypher, a database manipulation language, was used to show that we could retrieve *bad smells* in the Q&A process and visualize any version of the requirements list evolving through the processes.

Keywords: Requirements Elicitation · Q&A Process · Bad Smell · Version Control · Graph Database · Object Oriented Modeling.

1 Introduction

Requirements elicitation, which is the first activity in a development process, has significant effects on the quality of the developed software, and at the worst cases where its requirements specification of lower quality has been obtained, the developers might re-do the development activities. Usually, a requirements analyst gets an initial list of requirements from stakeholders at first, and then conducts question-and-answer (Q&A) sessions iteratively to incrementally make the initial list more complete. The Q&A sessions are significant to get a resulting requirements specification of higher quality. However, iterated Q&A sessions often contain some problems, e.g., vagueness of the timing when an analyst and stakeholders could bring their discussions to a close, no sufficient consensus of discussion conclusions between them, inconsistencies in the resulting requirements list, etc. As a result, the specification, the final product of a requirements

elicitation step, may contain the inconsistent requirements and/or unwanted ones, and mandatory requirements may be missing in the specification.

One of the ways to avoid this situation is 1) to keep the records of the activities of the Q&A sessions together with the track of evolving the requirements and 2) to identify the occurrences of so-called *bad smells*, a symptom possible to lead to these problems, by retrieving the stored records. To achieve the above two goals, in this paper we clarify bad smells included in Q&A activities[5], and we realize a database system storing Q&A activities tightly connecting to developed requirements list. We also design database queries to detect the occurrences of *bad smells*. We use a graph database system, Neo4j [10] to store Q&A activities and elicited requirements together. Cypher (a data manipulation language of Neo4j like SQL) is used to represent database queries not only to detect *bad smells* of the Q&A activities but also to control the versions of produced requirements lists, e.g., extract a certain version of the requirements list, differences between its two versions, etc. Our contribution is to 1) define *bad smells* for Q&A activities, 2) model Q&A activities to detect the occurrences of the defined *bad smells* and to realize the model by a graph database system Neo4j, and 3) demonstrate useful database queries for detecting the defined *bad smells*.

The rest of the paper is organized as follows. We first clarify our concrete problems to be addressed and then we refine the problems to *bad smells* and to a rough sketch of queries to detect them. Section 3 presents our data model for the graph database system so that we can realize our queries mentioned in the previous section. In Section 4, we pick up a couple of our designed queries and demonstrate their execution results. Our demonstration shows that our queries can work well for detecting bad smells and for controlling versions of requirements lists. Section 5 is for discussions on our contributions. Related work and concluding remarks are mentioned in Sections 6 and 7, respectively.

2 Our Approach

2.1 Problems in Q&A Activities to be Addressed

Q&A sessions for requirements elicitation may include the following characteristics that provide lower efficiency of development processes and lower quality of developed products.

1. None of an analyst and stakeholders can grasp the whole picture of the process. As a result, sometimes the analyst and stakeholders cannot decide where the end of the activities is.
2. There can be missing implementation of requirements. If the conclusions and consensus on Q&A sessions were neither maintained nor connected to resulting requirements, missing requirement items would occur.

[5] Q&A activities in a session are a sequence of someone's utterances and responses from the others.

3. There can be insufficient communication between an analyst and stakeholders. If this situation happens, opinions of a certain stakeholder may not be reflected in the final version of a requirements list.
4. An analyst and stakeholders cannot come to a consensus, to the end, or their consensus building can take longer time.
5. Multiple discussions on a requirement content on the different agenda topics allow the analyst to list up duplicated requirements that may be inconsistent to each other, and as a result the requirements list can include inconsistency.

2.2 Supporting Q&A Activities Using a Graph Database

To detect the occurrences of the problems mentioned in Section 2.1, we record Q&A activities, versions of a requirements list and their relationships in a database system, and by using database queries we automate the detection of their occurrences. An initial requirements list, which has been firstly provided by stakeholders, is stored to the database. An analyst or the stakeholders pick up one of the list items as a topic, and start a Q&A session. For example, the analyst may suggest the details of the picked-up topic and/or propose a new requirement related to the topic. The stakeholders may respond to the analyst's suggestion or proposal. The response allows the analyst to update the requirements list and to produce its newer version. The differences between the older list and the newer one should be identified in the database, i.e., our system should have the function of version control. These Q&A activities are structured and stored in the database, connecting to their related requirements items. During Q&A sessions, the analyst and the stakeholders can access to the database and can retrieve the symptoms of the problems (to be discussed in Section 2.3) by using queries represented with a database manipulation language. We use a graph database system Neo4j and its manipulation language Cypher to represent queries. The reasons why we use a graph database system are mainly from implementation view. Firstly, we can update the records of Q&A activities dynamically and flexibly, and secondly we can visualize the results of manipulating data so that the analyst and the stakeholders can understand the results easily at a glance. They are not always an expert of Neo4j or Cypher, so the understandability and the easiness of making queries are very important. We have developed pre-defined templates of queries to retrieve the problems in Q&A activities for each type of their symptoms.

2.3 Bad Smells — Symptoms of the Problems

Next, we should clarify the techniques to detect *bad smells*, a symptom which may cause the problems mentioned in Section 2.1, from the recorded Q&A activities are necessary. By using GQM (Goal-Question-Metric) technique [4], where we set "detecting the problems mentioned in Section 2.1" as Goals, *bad smells* of the Q&A activities as Questions (Are there bad smells?), and database queries to retrieve the occurrences of *bad smells* as the implementation of calculating Metrics. In our approach, Q&A activities and produced requirements are stored

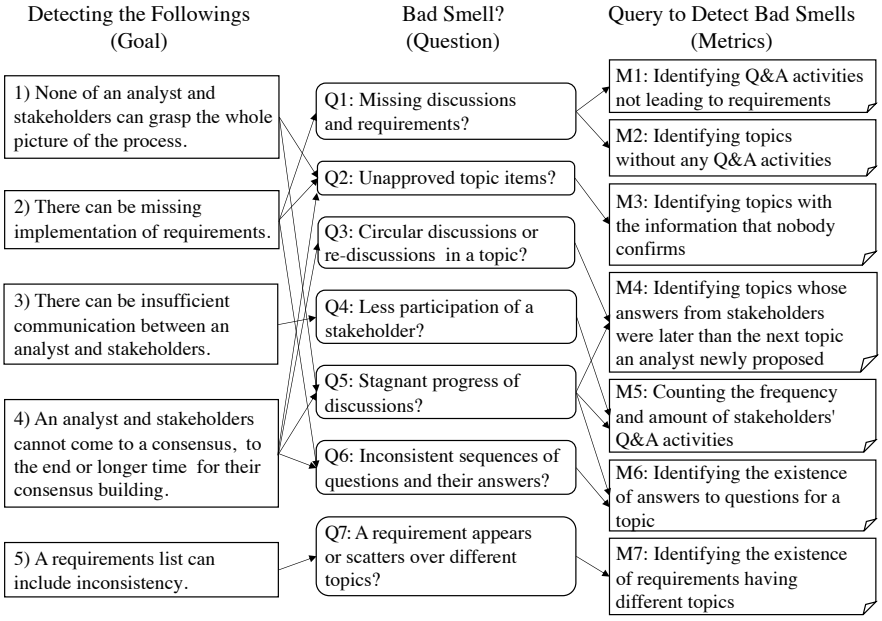


Fig. 1. GQM approach.

in a database system, and therefore the metrics to answer these questions can be defined and calculated with queries on the database system.

Figure 1 illustrates the application of GQM technique. In the example of this figure, one of our goals is to check if Q&A activities do not contain the problem "There can be missing implementation of requirements". To achieve it, we set the three questions "(Are there) Missing discussions and requirements?", "(Are there) Unapproved topic items?" and "(Are there) Inconsistent sequences of questions and their answers?". Consider the case of unapproved topic items as an example. Since an unapproved topic item cannot be usually included in a final requirements list, it will not be implemented in a final product. If the requirement on the unapproved topic item is mandatory to stakeholders, the resulting specification may lose a significant requirement. To find unapproved topic items in our database, we can attach to a discussed and then approved topic a link denoting "confirmed". Our query searches for the occurrences of the link "confirmed" for each discussed topic in the database.

Inconsistent sequences of questions and their answers may be the problem "There can be missing implementation of requirements" as follows. Consider the situation where none of the stakeholders answered to a question from the analyst or the situation where they did not answer to it until the analyst moved to the other new topics. In this situation, the analyst could not find the occurrences of logically consistent sequences of questions and their answers, and as a result, she or he would lose sight of the requirements to be implemented.

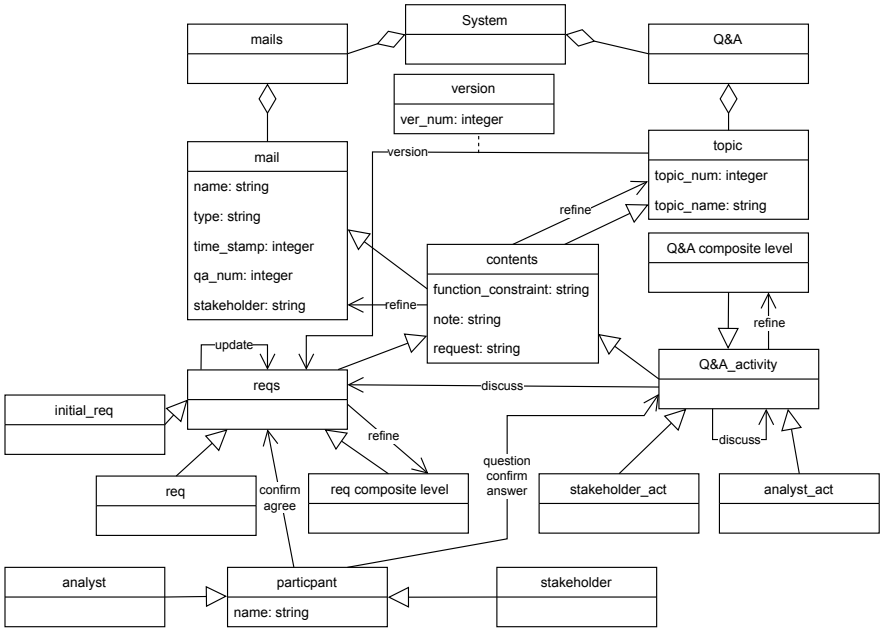


Fig. 2. Object-oriented model of Q&A processes for a graph database model.

2.4 Q&A Activities for Development a Library System

In this subsection, we mention briefly an example to explain our technique in successive sections. This example is a Q&A process to develop a library system, where a practitioner as a requirements analyst conducted it with a library user and a librarian as stakeholders. They communicated with each other by E-mail and through these Q&A activities the analyst gradually made complete a given initial list of requirements of a library system [15]. In this example, we had 43 mails, 7 initial requirements, and 40 requirements that were newly derived from the Q&A sessions.

3 Modeling Q&A Processes

3.1 Object-Oriented Model for Q&A Processes

To realize the queries mentioned in Section 2.3, we design an object-oriented model of Q&A processes for a graph database, and uses it as a kind of a schema of graph database systems, as shown in Fig. 2. One of the reasons why we used an object-oriented technique is that we can easily implement a graph database system from the resulting object-oriented model. It is simple how to transform the object model like Fig. 4 to the data of Neo4j. Basically, since classes, associations (including aggregations), and attributes directly corresponds to nodes,

edges, and properties of property graph models of Neo4j, the concrete data follows the form of the object model.

The model in the figure is rather an extendable framework by adding subclasses according to Q&A processes. For example, if a Q&A process has more refined types of stakeholders such as users and system administrators etc., we can add them as subclasses of "stakeholder" class. In the case of our library example, we have added "librarian" class and "(library) user" class.

We use "generalization" relationships (super-subclass relationships) to inherit attributes and associations to subclasses. For example, the class "mail" has an attribute "time_stamp", and its subclasses "contents" and "Q&A_activity" can also have it. Since "contents" is a subclass of "topic", the instances of "contents" can have the values of the attributes "topic_name" etc. "Contents" has the attributes "function_constraint", "request", and "notes", which is neither of them. The attribute "function_constraint" specifies whether the content is on a function or a constraint, and "request" can be for a product or for a Q&A session. This kind of usage of generalization is useful to reduce the types of nodes and edges of Neo4j, and as a result we can design simpler queries for retrieving bad smells.

In the model, we use *composite pattern* of Gang-Of-Four to represent a hierarchical nested structure. For example, the classes "reqs", denoting a requirement, and "req composite level" (a set of requirements) for representing the nested requirements, are a component and a composite respectively in a composite pattern. Note that we use the association labelled "refine" instead of aggregation relationship on applying a composite pattern, to differentiate the association "refine" from aggregation used in the non-nested structure. On Neo4j queries, we identify easily that the edges labelled with "refine" can lead to a nested structure. Aggregation can be considered essentially as a kind of association.

The contents of a "Q&A_activity" can also have a nested structure, and we also use a composite pattern for this part. There are some classes having both a generalization and an association such as "refine", except the usage of composite pattern. For example, there are a generalization and a "refine" association between "mail" and "contents". The "refine" association is intuitively considered as an aggregation and is used for pulling from a node of type "mail" its "contents" in retrieving bad smells in Neo4j. Similarly, the classes "topic" and "contents" have a generalization and a "refine" association. The body of a mail, contents, a Q&A activity or requirement is stored in the attribute "name" declared in "mail" class. "Topic" is a kind of a header to identify easily contents and also relates to contents with "refine" association.

A requirement "req" could have its version information itself as an attribute value, but rather we use a link between "reqs" and "topic" having a link attribute "ver_num" to make it easier to retrieve specific versions with links as edges in Neo4j. The class "reqs" has the subclass of initial requirements list "initial_req", which is an input of the Q&A process. The link "update" between "reqs" represents the evolution of requirements through Q&A processes, and we can reason the rationales of the evolution by tracing the link "discuss" from "Q&A_activity".

```
The mail sent from the analyst at 2020/5/6 15:40, whose qa_num is 4.
                              ...
2. Question on Registration of materials to and removal from the library
Does it mean
we register lendable materials to the library system, and
we remove lendable materials from the library system?
                              ...
```

Fig. 3. Example of a part of a mail.

An analyst or a stakeholder (a participant) performs an activity of "question", "answer", or "confirm" to a content of Q&A activity. It is specified with the type of associations "question", "answer" or "confirm" between the participant and her performed Q&A activity, as shown in Fig. 2.

The "discuss" association between "Q&A_activity" shows the relationship from a Q&A activity not only to a requirement but also to another Q&A activity. Suppose that a stakeholder asked a question to an analyst, but after that, the analyst asked another question to this stakeholder's question because the analyst could not understand well the stakeholder's question. The "discuss" association is useful to hold the relationship between these two Q&A activities.

Figures 3 and 4 show an example of a part of a mail for the development of a library system and its object model as the instance of Fig. 2. This mail was sent by the analyst at 15:40 on May 26th in 2020 and had the question-answer number (qa_num) of 4. The time stamp stands for the date when the mail was sent, and we had a 12 digits number "202005261540" as its time stamp. The mail consisted of several contents, and this example shown in the figure is its second part. The topic of the example has a content whose name (topic_name) is "Registration of materials to and removal from the library", whose topic number (topic_num) is 2 as shown in the heading part of the content. The part also tells us that the content is a "question" and refers not to a constraint but to a function of the Library System. Thus, we put the attribute values "question" and "function" as "type" and "function_constraint", respectively. Since this example is a question from the analyst, it is also an instance of "analyst_act" class whose super class is "Q&A_activity".

When the topic refers to a specific version of requirements lists, the reference is specified with a link from the topic[6] to the corresponding requirement. This example refers to the element of the initial requirements list "Registration of books to and removal from the library". The analyst tried to extend the interpretation of word "books" to "lendable materials" in this question. We used as a version number the number derived from the date when the mail was sent, and attached the number 20200526 as its version number in the link to the requirement.

We manually extracted data from 43 E-mails and structured them following this model with ten hours or less. As will be discussed in Section 7, some automated techniques are necessary.

[6] In this example, a link from the Q&A_activity whose class is a subclass of "topic".

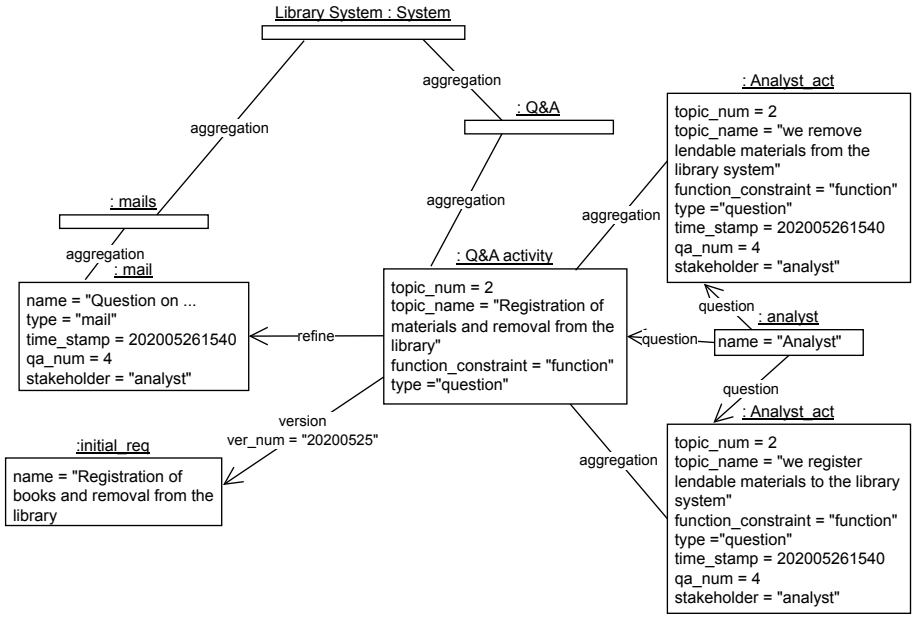


Fig. 4. Object model of the example of Fig. 3.

3.2 Neo4j and Cypher

Figure 5 shows an example of concrete data stored in the graph database Neo4j. It is simple how to transform the object model like Fig. 4 to the data of Neo4j. Basically, since classes, associations including aggregations and attributes directly corresponds to nodes, edges, and properties of property graph models of Neo4j, the concrete data follows the form of the object model. Note that the browser of Neo4j colors nodes and edges to distinguish the types of the nodes and the edges. Instead, for accessibility, we added stereo types denoting the type names in the figures in this paper as needed.

The data manipulation language of Neo4j is Cypher, which is based on SQL. We use `CREATE` statements to create nodes and `MATCH` ones to retrieve subgraphs satisfying a certain condition.

Query 1. Data retrieval.

```
1  MATCH (n1:stakeholder)-[:confirm]->
2      (n2:analyst_act{name:"we register lendable materials"})
3      RETURN n1, n2;
```

The above example of a `MATCH` statement outputs the nodes having specific values as their properties and connecting an edge labelled `confirm`. We show a part of templates of Cypher queries to detect bad smells in the next sections.

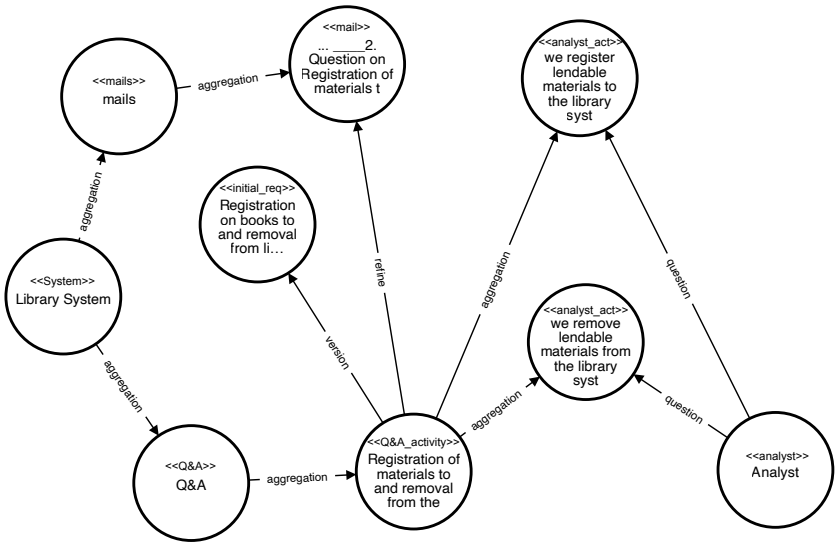


Fig. 5. Graph model of the example of Fig. 3.

4 Detecting Bad Smells and Controlling Versions using Cypher Queries

4.1 Detecting Missing Discussions and Requirements (Q1 in Fig. 1)

To detect Q1: missing discussions and requirements, we identify 1) the topics that their participants have not discussed (corresponding to M2 in Fig. 1) and 2) the requirement items that had been discussed but are not put in the requirements list (M1 in Fig. 1). As shown in Fig. 1, we have two types of queries and one of them is shown as follows.

Query 2. Identifying Q&A activities not leading to requirements (M1 in Fig. 1).

```
1  MATCH (n:analyst_act)-[:discuss]->(m)
2      WHERE NOT (m:req)
3      RETURN n;
```

This query template is used for M1: identifying Q&A activities not leading to requirements. Usually, an analyst starts discussions, maybe utters a question including a proposal, and after some Q&A activities, a requirement related to them is put in the list. However, if the requirement was not put in the list, it could result in a missing one. By using Query 2, we retrieve the nodes of `analyst_act` (analyst's activity) having no edges outgoing to `req` labelled with `discuss`. We can use `stakeholder_act` instead of `analyst_act` as the type of graph node `n`. Figure 6 shows the result of executing Query 2. In the figure, we can see 14 Q&A activities that have not been led to any requirement yet. For example, the graph node "Publisher" (analyst's proposal that "Publisher"

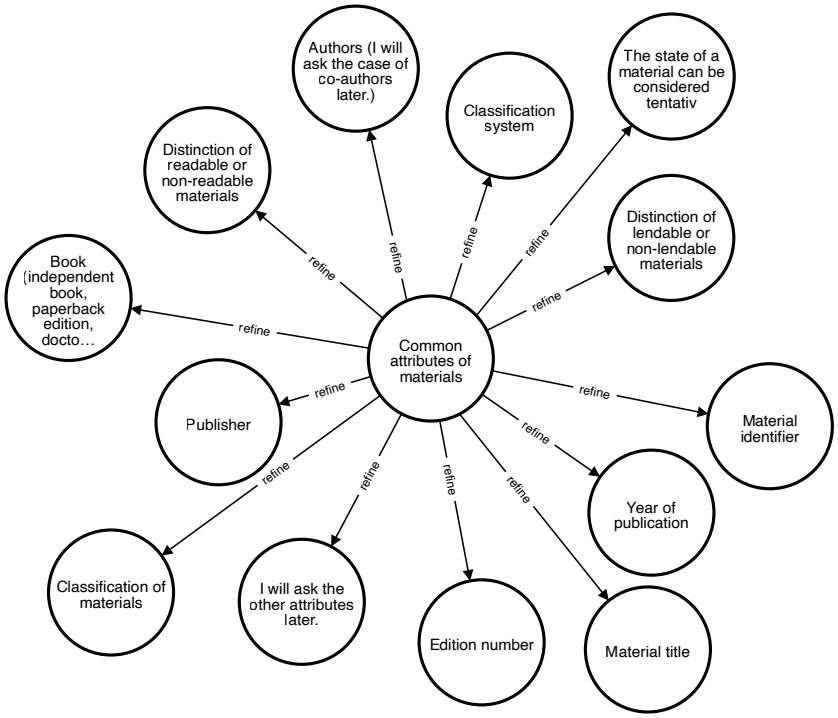


Fig. 6. Result of identifying Q&A activities not leading to requirements.

information should be included in material's attribute data) is proposed to be discussed, but it does not appear as a requirement yet.

4.2 Unapproved Topic Items (Q2 in Fig. 1)

To detect the bad smell "Q2: Unapproved topic items?", we retrieve the graph nodes of requirements (`req`) that have no edges of "confirm" from any stakeholder. Thus, we have the following query template:

Query 3. Identify topics with the information that nobody confirms (M3 in Fig. 1).

```
1  MATCH (g:analyst_act{type:"confirm"})-[:discuss]->(f:req)
2            <-[:discuss]-(h:stakeholder_act)
3      WHERE NOT (h)-[:confirm]->(f)
4      RETURN g, h, f;
```

The matching pattern in a `MATCH` phrase of Query 3 is to search for the Q&A activity where an analyst asks stakeholders for the confirmation of a requirement and the stakeholders discuss it. The `WHERE` clause specifies the stakeholders have not confirmed it yet. That is to say, the requirement is put in the requirement list without any confirmation from the stakeholders. If you specify a certain stakeholder, e.g., librarian, you can add the condition `h.name="Librarian"` in the `WHERE` clause.

4.3 Circular Discussions or Re-Discussions in a Topic (Q3 in Fig. 1)

To detect the occurrences of Q3: circular discussions or re-discussions in a topic, we focus on time sequences of Q&A activities. Our database system holds the information of a time stamp when a Q&A activity was uttered and of temporal ordering on pairs of questions and their answers. As for the latter information, the property "qa_num" (question & answer number) of the model shown in Fig. 2 represents the order of Q&A activities for a topic.

Consider the following scenario. *An analyst asked stakeholders "common attributes of materials" on the topic "Registration of materials to and removal from the library" and its activity was numbered with 7 as "qa_num" property. If the stakeholders answered it immediately, their answer was numbered with the same number. However, actually they did not do anything on the analyst's question and they kept silence. Therefore, the analyst was forced to conduct the other different Q&A activities. Since the analyst conducted the different activities, the "qa_num" was advancing. When the stakeholders finally answered, its "qa_num" was 25. From the view of time sequence, a question and its answer were not successive but the other Q&As was disrupted between them. That is to say, we identify topics or Q&A activities whose answers from stakeholders were later than the next topic an analyst newly proposed. The analyst considered that the discussion on "qa_num" 7 was finished, but the late answer with "qa_num" 25 could cause other discussions on the same topic.* Thus, late answers like this scenario may cause to circular or re-work discussions.

We have a query template to identify them by using "qa_num" as follows.

Query 4. Identifying topics whose answers from stakeholders were later than the next topic an analyst newly proposed (M4 in Fig. 1).

```
1  MATCH (n)-[:discuss]->(m{type:"answer"})
2      WHERE (n.qa_num + DELTA < m.qa_num)
3      RETURN (n)-[:discuss]->(m);
```

The parameter `DELTA` stands for the scope where we retrieve late answers. We can specify the types of graph nodes `n` and `m` as `analyst_act` and `stakeholder_act` and vice versa. Figure 7 shows the result with `DELTA` = 3, `analyst_act` as `n` and `stakeholder_act` as `m`[7]. It demonstrated the above scenario where the analyst asked "Common attributes of materials" (Are there any other common attributes of materials?) and where the late answer from the stakeholder "User" contained the proposals of new attributes such as ISBN and Purchase date. The analyst should start re-discussions on "Common attributes of materials" even if she or he considered that its discussions was finished.

4.4 Version Control

The functions of version control are not directly related to detecting bad smells of Q&A activities, but they are necessary to monitor the progress of the re-

[7] When we click a node on a Neo4j browser, we can see the content of its property values. To make this figure more visible, we paste the property values of the node "Is the following information unnecessary as ...".

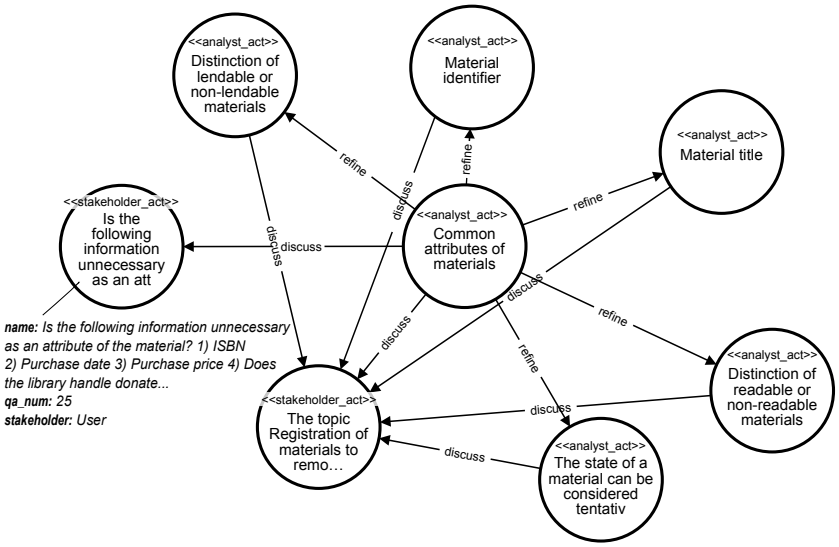


Fig. 7. Circular discussions or re-discussions.

quirements elicitation process. In addition, since the analyst can access to the evolution of the requirements list and can check the rationales why some requirements in the list had been changed and evolved by accessing the linked Q&A activities, it could be useful to avoid circular Q&A activities on the evolved requirements and to capture milestones throughout a Q&A process. In this paper, we list up two types of queries; one is to check-out a certain version of requirements shown in Query 5, and the other is to extract the differences (diffs or delta) between two versions shown in Query 6. The latter is useful to capture the evolution of requirements and its rationales.

Query 5. Check-out.

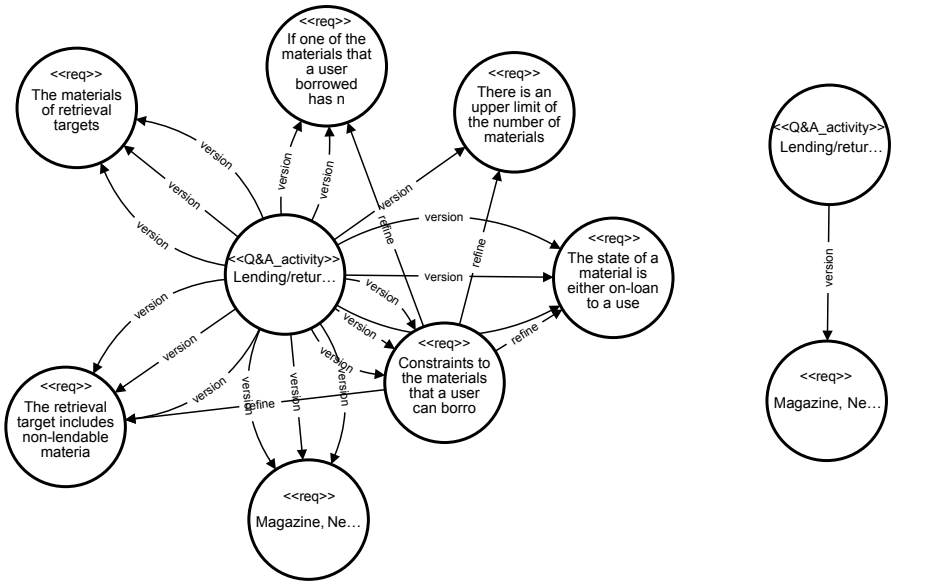
```
1  MATCH (n)-[:version{ver_num:XXX}]->(m:req)
2      RETURN n, m;
```

Query 5 is used to extract requirements of version number "`XXX`". The node `m` is the extracted requirement, and we can see the content of the requirement by clicking the node on the Neo4j browser.

Query 6. Difference extraction.

```
1  MATCH (k)-[:version{ver_num:YYY}]->(l:req)
2      WHERE NOT (k)-[:version{ver_num:XXX}]->(l)
3      RETURN k, l;
```

By using Query 6, we retrieve the requirements which the version "`XXX`" has not but "`YYY`" does, i.e., $Req(\texttt{YYY}) \backslash Req(\texttt{XXX})$ where $Req(\texttt{NNN}) = \{r \mid r \text{ is a requirement of version number "}\texttt{NNN}\text{"}\}$. Figure 8 shows the result of the differences between the versions 20200608 and 20220626. In the left side of the figure, the nodes of



(a) Newly added requirements.

(b) Deleted requirement.

Fig. 8. Differences between the versions 20200608 and 20200626.

`req` denote the requirements that had been newly added to the newer version 20200626, i.e., the requirements that had not been in the older version 20200608. It resulted from setting 20200628 and 20200608 to `YYY` and `XXX` in Query 6, respectively. On the other hand, Fig. 8(b) shows the requirement that had been deleted in the version 20200626, but that was in the version 20200608. We can retrieve the deleted requirements in the newer version by setting the older one to `YYY` and the newer to `XXX` in Query 6. Note that the `req` node "Magazine, Ne ..." seems to be the same in Figs. 8(a) and 8(b), but their contents were different. Both of the nodes presented the scope of lendable materials and in the newer version, and it was updated and clarified. In Fig. 8, three occurrences of `version` edge between `Q&A activity` and `req` can be found. The Neo4j browser outputs all of the edges between retrieved nodes. One of the three is the version 20200626, and the other two show that the requirements remained in the latter two versions.

5 Discussion

For our Neo4j database system of Library system example, we have designed and executed the queries listed up in Fig. 1. By analyzing the retrieval results, all of the queries except three could identify precisely the subgraphs that we want, which could be used for finding bad smells. These three cases are 1) M1: identifying discussions not leading to requirements, 2) M4: identifying topics

whose answers from stakeholders were later than the next topic an analyst newly proposed, and 3) M7: identifying the existence of requirements having different topics. With them we have retrieved unnecessary subgraphs in addition to the subgraphs that we wanted. That is to say, these queries output *false positives*. The reasons why these queries output the unnecessary graphs can be listed up as follows.

- To match the analyst's specific Q&A activities, we might sometimes use weak conditions. In the example of Query 4, we used 3 as the threshold `DELTA`. However, our query found out the answer delayed later than 3 but before the next topic started, and indicated an occurrence of *circular or re-work discussions* bad smell. More sophisticated decision, e.g., comparing topics, may be necessary.
- We used as requirements the sentences appearing in the property "name" values in mails. Although our requirements analyst was a veteran, he sometimes used different words even to express the same meaning or concepts. As a result, we could miss some of the mails including the same topic. This is not a problem on our queries, but on how to construct a Neo4j database.

Neo4j is not the only one system to implement our goal, and other techniques such as simply Excel plus its lining mechanism can realize it. However, comprehensive user interface and the query language Cypher to define bad smells flexibly allowed us to take Neo4j. In Neo4j, we can grasp easily the information connected to each other from bird's eye view, and especially we can capture progress of Q&A sessions. In contrast, for the hierarchical structures of requirement sentences and Q&A activities, in our graph database we should trace nodes and edges to recognize their hierarchical structures. So, it is not so easy to grasp these hierarchical structures at a glance and it is a demerit of using a graph database.

6 Related Work

We can find several studies to support interviews and meetings among analysts and stakeholders. Until the end of 1990s, the application of Searle's speech act theory has often been studied to navigate speakers in Q&A sessions [7, 11]. Inquiry Cycle Model [11] is to capture discussions between analysts and stakeholders with Q&A cycles which are patterned with a set of speech acts such as Question, Answer, and Reason. They are navigated following the patterned cycle and their speech acts in the cycle trigger to update a requirements specification. The idea of specific speech acts connecting to evolution of a requirements specification is similar to our model of Neo4j. However, including Inquiry Cycle Model, this category of studies had the idea where analysts and stakeholders are forced into predefined patterns of discussions to avoid *bad smells*. On the other hand, our approach does not force their activities, and instead detect *bad smells* to improve their activities. Since requirements elicitation is one of the most creative tasks of human, over enforcement to its Q&A activities may lead

to a specification of lower quality. Rather, we can use the results of this category as complementary ones. In addition, the studies of this category did not mention explicitly the idea nor the definition of *bad smells*, which this paper did. Similarly, interview patterns of expert analysts [2, 8, 9] can be useful to enhance our approach, especially creating a graph database system. In addition, there are studies of analyzing interviews between analysts and stakeholders to detect ambiguity during the interviews [14]. These results can be applied to design more sophisticated queries to detect bad smells on ambiguity basis. As for combining Q&A activities with artifacts, we used links between Q&A activities and their resulting requirements.

There are several studies on bad smells in requirements engineering [6, 12], but many of them are for products, i.e., elicited requirements themselves. This study focuses not on products but on processes to elicit requirements. Ambriola and Gervasi [1] discussed process metrics of requirements analysis using stability of requirements and the efforts of reworks on a part of documents. These metrics are useful to explore new types of metrics to detect bad smells.

7 Conclusion and Future Work

This paper presented an integrated technique of detecting bad smells in Q&A activities between a requirements analyst and stakeholders during requirements elicitation and of controlling the versions of elicited requirements on a graph database Neo4j. The analyst and the stakeholders can access to it using queries in order to detect *bad smells*, symptoms to cause Q&A processes and requirements specifications of low quality, during Q&A sessions. We picked up E-mail bases Q&A activities to develop a library system and constructed a Neo4j database. We designed and executed Cypher queries to detect bad smells and to extract diffs of versions. This is just a proof-of-concept of our technique. The future research agendas, which were obtained from our case study, can be listed up as follows.

- **Tailoring a bad smell catalog:** The bad smells in Fig. 1 resulted from a case study of an expert's E-mail interview, so we can add more new bad smells. To discover them, the analytic results in [3] are very useful.
- **Structuring Q&A activity records for a graph database system:** In this study, we extracted data from a large volume of E-mails and stored them in the database system, with much more human efforts. The technique to summarize interviews like [13] can be applied to reduce human efforts. If we take E-mail based Q&A sessions, we will apply generative AI techniques to automatically classify the texts of E-mails and extract information necessary to generate a graph database system. The annotation tool such as AnnoteREI [5] can support structuring Q&A activities following our model of Fig. 2.

Acknowledgements This work was partly supported by JSPS Grants-in-Aid for Scientific Research No. JP21K11823 and Nanzan University Pache Research Subsidy I–A–2 for the 2024 academic year.

References

1. Ambriola, V., Gervasi, V.: Process metrics for requirements analysis. In: Proc. 7th European Wprkshop on Software Process Technology. pp. 90–95 (2000)
2. Ann Scheinholtz, L., Wilmont, I.: Interview patterns for requirements elicitation. In: Proc. 17th International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 72–77 (2011)
3. Bano, M., Zowghi, D., Ferrari, A., Spoletini, P., Donati, B.: Learning from mistakes: An empirical study of elicitation interviews performed by novices. In: Proc. 26th IEEE International Requirements Engineering Conference. pp. 182–193 (2018)
4. Basili, V.: Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Tech. Rep. UMIACS-TR-92-96, University of Maryland (1992)
5. Debnath, S., Subramanian, S.: AnnoteREI! A tool for transcribing and annotating requirements elicitation interviews. In: Proc. 30th IEEE International Requirements Engineering Conference. pp. 255–256 (2022)
6. Femmer, H., Fernández, D.M., Jürgens, E., Klose, M., Zimmer, I., Zimmer, J.: Rapid requirements checks with requirements smells: Two case studies. In: Proc. 1st International Workshop on Rapid Continuous Software Engineering. pp. 10–19 (2014)
7. Kato, J., Komiya, S., Saeki, M., Ohnishi, A., Nagata, M., Yamamoto, S., Horai, H.: A model for navigating interview processes in requirements elicitation. In: Proc. 8th Asia-Pacific Software Engineering Conference. pp. 141–148 (2001)
8. Malviya, S., Vierhauser, M., Cleland-Huang, J., Ghaisas, S.: What questions do requirements engineers ask? In: Proc. 25th IEEE International Requirements Engineering Conference. pp. 100–109 (2017)
9. Mohedas, I., Daly, S., Loweth, R., Huynh, L., Cravens, G., Sienko, K.: The use of recommended interviewing practices by novice engineering designers to elicit information during requirements development. Design Science **8**, e16 (2022)
10. Neo4j: https://neo4j.com/
11. Potts, C., Takahashi, K., Anton, A.: Inquiry-Based Requirements Analysis. IEEE Software **11**(2), 21–32 (1994)
12. Seki, Y., Hayashi, S., Saeki, M.: Detecting bad smells in use case descriptions. In: Proc. 27th IEEE International Requirements Engineering Conference. pp. 98–108 (2019)
13. Spijkman, T., de Bondt, X., Dalpiaz, F., Brinkkemper, S.: Summarization of elicitation conversations to locate requirements-relevant information. In: Proc. 29th International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 122–139 (2023)
14. Spoletini, P., Ferrari, A., Bano, M., Zowghi, D., Gnesi, S.: Interview review: An empirical study on detecting ambiguities in requirements elicitation interviews. In: Proc. 24th International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 101–118 (2018)
15. Wing, J.M.: A study of 12 specifications of the library problem. IEEE Software **5**(4), 66–76 (1988)