PAPER Special Section on Knowledge-Based Software Engineering

Cataloging Bad Smells in Use Case Descriptions and Automating Their Detection*

Yotaro SEKI[†], ***Nonmember***, **Shinpei HAYASHI**[†a)], ***and*** **Motoshi SAEKI**[††b)], ***Members***

SUMMARY Use case modeling is popular to represent the functionality of the system to be developed, and it consists of two parts: a use case diagram and use case descriptions. Use case descriptions are structured text written in natural language, and the usage of natural language can lead to poor descriptions such as ambiguous, inconsistent, and/or incomplete descriptions. Poor descriptions lead to missing requirements and eliciting incorrect requirements as well as less comprehensiveness of the produced use case model. This paper proposes a technique to automate detecting bad smells of use case descriptions, i.e., symptoms of poor descriptions. At first, to clarify bad smells, we analyzed existing use case models to discover poor use case descriptions concretely and developed the list of bad smells, i.e., a catalog of bad smells. Some of the bad smells can be refined into measures using the Goal-Question-Metric paradigm to automate their detection. The main contributions of this paper are the developed catalog of bad smells and the automated detection of these bad smells. We have implemented an automated smell detector for 22 bad smells at first and assessed its usefulness by an experiment. As a result, the first version of our tool got a precision ratio of 0.591 and a recall ratio of 0.981. Through evaluating our catalog and the automated tool, we found additional six bad smells and two metrics. Then, we obtained the precision of 0.596 and the recall of 1.000 by our final version of the automated tool.

key words: *use case descriptions, smell detection*

1. Introduction

Use case modeling is one of the popular techniques to elicit requirements to business processes, information systems, and software (simply, software hereafter), and is being made into practice [2]. Software developers can apply use case models to validate software design to requirements, to make test plans and development schedules, etc., and use case modeling can have wide variety of applications.

A use case model consists of two parts: use case diagram and use case description. Use case descriptions are structured text written in natural language, and the usage of natural language can lead to poor descriptions such as ambiguous, inconsistent and/or incomplete descriptions. Poor descriptions lead to missing requirements and eliciting incorrect requirements as well as less comprehensiveness of produced use case models. In fact, many use case models of lower quality have been produced [3]. Thus, it is significant to detect poor use case descriptions during the development of a use case model.

There have been developed several guidelines to compose use case descriptions of higher quality and checklists to detect problematic parts in them. For example, Phalp *et al*. developed a checklist called 7C's, which was specified in natural language [4], and Törner *et al.* provided questionnaires as their quality checklists [5]. However, these guidelines can be applied manually only, and they allow us to miss problematic parts in use case descriptions. Manual decision making of a poor description is subjective, and the results may depend on the requirements analysts. Furthermore, manual checking is a time-consuming task for the analysts [4].

This paper proposes a technique to automate detecting *bad smells* of use case descriptions, symptoms of poor descriptions. First of all, for use case descriptions, we define *bad smells*, where the concept of bad smells has been established mainly for source code as their surface characteristics that might cause deep problems [6]. We make a catalog of bad smells from two views: their characteristics and levels of granularity (scope) of occurring bad smells. Note that bad smells are mainly not *bugs* but indicators of poor descriptions that may cause some serious problems in future steps of a software development project. To make the catalog, we collected 30 use case descriptions of various problem domains using the Internet or other resources such as textbooks and tried to find *bad smells* out of them. As a result, we got 54 bad smells. Next, by applying Goal-Question-Metric (GQM) paradigm [7], we got metrics to detect 22 of the 54 bad smells and developed an automated tool based on these metrics. Through evaluating our catalog and the automated tool, we found additional six bad smells and two metrics. Finally, we got the precision ratio of 0.596 and recall ratio of 1.00 to detect bad smells by our final version of the automated tool.

The rest of this paper is organized as follows. We first present the overview of our approach in the next section. In Sect. 3, we present the developed catalog of bad smells. We show the application of the GQM approach, the set of the derived metrics, and the tool implementation in Sect. 4. Section 5 presents the experimental evaluations of the catalog and the automated tool. In this section, we used the other

Manuscript received May 10, 2021.
Manuscript revised October 23, 2021.
Manuscript publicized January 6, 2022.

[†]The authors are with School of Computing, Tokyo Institute of Technology, Tokyo, 152–8550 Japan.
[††]The author is with Department of Software Engineering, Faculty of Science and Technology, Nanzan University, Nagoya-shi, 466–8673 Japan.

*By adding more explanations of bad smells etc., this paper is extended based on [1], which appeared in proceedings of the 27th IEEE International Requirements Engineering Conference, © 2019 IEEE.

a) E-mail: hayashi@c.titech.ac.jp
b) E-mail: saekimot@gmail.com

DOI: 10.1587/transinf.2021KBP0008

eight use case descriptions different from the 30 descriptions used for developing the catalog and the metrics. Sections 6 and 7 are for related work and concluding remarks.

Note that this paper is an extended version of [1], by emphasizing the detailed explanation of our smell catalog in Sects. 3 and 5, including what led us to the proposal, and by making deeper experimental analysis in Sect. 5.

2. Approach Overview

In our approach, existing use case descriptions were analyzed, and bad smells in use case descriptions and metrics for automatically detecting them were derived. The overview of our approach is shown in Fig. 1. First, we collected examples of existing use case descriptions from existing resources such as via the Internet or textbooks. Next, one of the authors identified the problems in the descriptions and abstracted them to extract bad smell types and compose a smell catalog. A catalog of bad smells may be useful to watch bad smells with a bird's eye view, as well as to use as a checklist for human analysts. In particular, we do not think that any bad smell can be automated to detect, and we can understand which bad smells can be automated to detect with the existing technology that we can get easily. These are reasons why we tried to make a catalog at the first stage of this research.

Subsequently, we applied the GQM paradigm [7] and derived metrics to identify the obtained smells. The derived metrics were machine-understandable so that an automated smell detector was implemented.

Note that this work was done in Japanese. We collected use case descriptions written in Japanese, and the smell identification and evaluation were conducted by native speakers of Japanese. This decision benefited us to better identify smells from many use case descriptions. All the example use case descriptions in this paper were originally written in Japanese and were directly translated into English.

The details of each study will be explained in the subsequent sections.

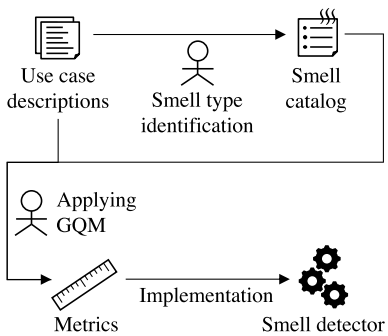


Fig. 1 Overview of our approach.

3. Defining Bad Smells and Their Catalog

3.1 Collecting Examples

Most samples of use case descriptions that we used for this study were retrieved using web searches with the query "use case descriptions." We have used not only the normal Google search but Google Image search to find images representing a use case description. We have also added a specific term of a domain such as "welfare" as an additional query to find domain-specific use case descriptions. We continued the search until we collected enough samples that met the following two conditions:

1. those written in natural language so that people can understand them, and
2. those contents are separated by sections so that the location where smells are occurring can be easily pointed out.

In addition to the found samples via a web search, we also used the descriptions found in research papers and textbooks that we have already known. Finally, we have gathered 38 sample use case descriptions in total, and we used 30 of the 38 to define bad smells and their catalog in Sect. 3. The other eight samples will be used for the evaluation of our catalog and smell detector in Sect. 5. These use case descriptions were chosen to cover a wide range of domains such as web application, health care, or welfare. An excerpt list of samples is shown in Table 1.

Figure 2 shows an example of the collected use case descriptions. This use case description is for searching products, and its contents are separated into multiple sections: Name, Basic Flow, and Alternate Flows. We set our target use case descriptions in abstract syntax level, referring to the metamodel of Misbhauddin *et al.* [8]. As illustrated in

Table 1 Samples of use case descriptions

Name	Domain	Source
Move piece on board	Game	Search
Print elderly relief call application form	Welfare	Search
Create new blog account	Web	Textbook
Search for product	Shopping	Image Search

Name Search for product
Basic flow:

1. User inputs a query in the search page and presses the "Search" button.
2. System searches the products that match the query.
3. System shows the found products on the result page.

Alternate flows:

A1 If the length of the query is shorter than 3.
A1.1 System shows the search page again and shows "Provide a query whose length is more than 2."

Fig. 2 Example of a use case description.

Fig. 2, it has a use case name, an actor list, a basic flow, alternate flows, exception flows, a precondition list, and a postcondition one. A description section where the overview and the aims of the use case are described, which does not appear in Fig. 2, may appear. The actor list consists of actors, i.e., words and phrases. The description section, the precondition and the postcondition list are a set of sentences. The flows (a basic flow, alternate and exception ones) also consist of sentences, but unlike the description section and the conditions (preconditions and postconditions), all of the sentences should be ordered, e.g., labeled with numbers. In the case of alternate flows and exception ones, conditions when alternatives and exceptions can occur should be described in header parts of the flows.

3.2 Deriving Smells

One of the authors read the collected use case descriptions carefully and identified their problems together with their reasons. Then, smell types were derived according to the similarity of the reasons attached to the problems. As a result, 247 problems were totally identified in the 30 use case descriptions, and 54 smell types were derived from them. All the smell types are listed in Table 2 according to the classification criteria, which will be explained in Sect. 3.3.

Some bad smells in Table 2 are explained below.

- S_4 (Unclear Feasibility): In the case where a sentence S denoting an action A has some assumptions C for its execution, e.g., a branch condition in a conditional branch to execute A such as "If C, A", and where C are too ambiguous to decide its truth value, the sentence S could be infeasible although the action A has no problems.
- S_{16} (Flow Does Not Meet Precondition), S_{17} (Postcondition Not Satisfied): S_{16} means that preconditions do not hold at starting the execution of a flow. S_{17} is for postconditions, and it means that the postconditions cannot hold at the end of the execution.
- S_{19} (Behavior Ignores Condition): Although some conditions C should hold to execute a sentence S denoting actions, C do not hold at starting the execution of S. Suppose that there is a sentence "A customer selects goods from a commodity list" as an example. To execute this example sentence, it is necessary for the customer to already have "commodity list", which can be considered as a mandatory input to the action denoted by the example sentence. If neither preconditions are referring to "commodity list" nor sentences are producing "commodity list" before the example sentence, the execution of the example sentence intends to ignore the non-existence of "commodity list".
- S_{20} (Multiple Situations): A "situation" denotes the external environment that uses a use case. In the example of E-commerce, suppose that we have a sentence "A customer specifies desired goods with web or a shop front" in a use case. We have two different situations: 1) at a web system and 2) at a shop front. The execution of the use case is different whether the customer uses a web system or she/he goes to a shop front. It is preferable to divide this use case into two for each situation.
- S_{21} (Multiple Roles of an Actor), S_{22} (Multiple Actors of a Role): These bad smells are related to the "role" of actors. There is an actor name playing several different roles (S_{21}) and several different actor names playing the same role (S_{22}). For example, we have an actor name "responsible person", but we use this word as a person playing two different roles of "a person in charge of a shop" and "a person in charge of a web system". Although there can be a case where the same person plays both of these two roles at the same time in a real setting, it is recommended that two different actor names are used to avoid confusion. This example is regarded as having a bad smell of S_{21}.
- S_{28} (Relatively Over-Qualified Sentence), S_{29} (Relatively Under-Qualified Sentence): The sentence includes relatively too many occurrences of modifiers (S_{28}) or too few modifiers (S_{29}).
- S_{30} (Less Qualified Word in Following Sentences), S_{31} (Less Qualified Word in Above Sentences): We have an occurrence of a word W with a modifier M, e.g., "W of M", in a sentence S. S_{30} is the case where the word W alone appears in the sentences below S in spite that the word W is used with the same meaning as "W of M". Suppose that the word "user information" appears in the first sentence in a basic flow and the word "information" does in the third sentence. "User information" and "information" denotes the same meaning, i.e., user information. "Information" in the third sentence is less qualified because the modifier "user" is missing. We consider that the granularities of "user information" and "information" are different in the sense that "user information" is the refinement of "information". The usage of the word "information" in the third sentence leads to the bad smell S_{30}. For S_{31}, let us consider another example where the word "outlet" appears in the first sentence of a flow and the word "outlet of ATM" does in the third sentence. In this case, the word "outlet" is less qualified in the first sentence, but it is qualified and refined in the following sentences of the same flow. The bad smell S_{31} is on the word "outlet" in the first sentence.
- S_{36} (Over-Qualified Word): This bad smell is the case where the word W has the modifier M, but the meaning of W is the same as or is included in that of M. M is not qualified to W, and so it is useless.
- S_{37} (Non-Standalone Use Case): In this bad smell, it is considered that a use case A uses another use case B in the use case description of A, but B is lacking. The use case A is not standalone in the sense that it cannot be executed without the use case B. The smell is on the use case A to indicate that some use cases are lacking.

Table 2 Bad smells in use case descriptions

- $C_{1,2}$: ⟨***Ambiguity***, ***Section***⟩
  - S_1: Unordered Flow$^{\diamond\bullet}$
  - S_2: Origin-Free Exception Flow$^{\diamond\bullet}$
  - S_3: Origin-Free Alternate Flow$^{\diamond\bullet}$
- $C_{1,4}$: ⟨***Ambiguity***, ***Sentence***⟩
  - S_4: Unclear Feasibility
  - S_5: Origin-Free Operation Result
  - S_6: Sentence Interpretable as Multiple Meanings
- $C_{1,5}$: ⟨***Ambiguity***, ***Word***⟩
  - S_7: Pronoun$^{\diamond}$
  - S_8: Omitted Word
  - S_9: Using the Word "Actor"$^{\diamond}$
  - S_{10}: Undeclared Main Actor
  - S_{11}: Different Concepts by Same Word
  - S_{12}: Omitted Attribute
- $C_{2,2}$: ⟨***Incorrectness***, ***Section***⟩
  - S_{13}: Name Does Not Explain Content
  - S_{14}: Too Weak or Strong Condition
  - S_{15}: Much or Less Actors
- $C_{2,3}$: ⟨***Incorrectness***, ***Flow***⟩
  - S_{16}: Flow Does Not Meet Precondition$^{\bullet}$
  - S_{17}: Postcondition Not Satisfied$^{\bullet}$
  - S_{18}: Contradicted Sentences$^{\bullet}$
- $C_{2,4}$: ⟨***Incorrectness***, ***Sentence***⟩
  - S_{19}: Behavior Ignores Condition$^{\bullet}$
- $C_{3,1}$: ⟨***Granularity***, ***Usecase***⟩
  - S_{20}: Multiple Situations
- $C_{3,2}$: ⟨***Granularity***, ***Section***⟩
  - S_{21}: Multiple Roles of an Actor
  - S_{22}: Multiple Actors of a Role
- $C_{3,2}$: ⟨***Granularity***, ***Flow***⟩
  - S_{23}: Multiple Exception Flows at an Exception Branch Condition$^{\diamond *}$
  - S_{24}: Multiple Alternate Flows at an Alternate Branch Condition$^{\diamond *}$
- $C_{3,4}$: ⟨***Granularity***, ***Sentence***⟩
  - S_{25}: Long Sentence$^{\diamond}$
  - S_{26}: Short Sentence$^{\diamond}$
  - S_{27}: Sentence with Multiple Actions$^{\diamond}$
  - S_{28}: Relatively Over-Qualified Sentence$^{\diamond}$
  - S_{29}: Relatively Under-Qualified Sentence$^{\diamond}$
- $C_{3,5}$: ⟨***Granularity***, ***Word***⟩
  - S_{30}: Less Qualified Word in Following Sentences
  - S_{31}: Less Qualified Word in Above Sentences
- $C_{4,3}$: ⟨***Redundancy***, ***Flow***⟩
  - S_{32}: Multiple Flows with the Same Role*
  - S_{33}: Flow Unrelated to Postcondition*
  - S_{34}: Conditional Flow*
- $C_{4,4}$: ⟨***Redundancy***, ***Sentence***⟩
  - S_{35}: Repeating the Same Noun$^{\diamond}$
- $C_{4,5}$: ⟨***Redundancy***, ***Word***⟩
  - S_{36}: Over-Qualified Word
- $C_{5,1}$: ⟨***Lack***, ***Usecase***⟩
  - S_{37}: Non-Standalone Use Case
- $C_{5,2}$: ⟨***Lack***, ***Section***⟩
  - S_{38}: Missing Actor Section$^{\diamond}$
  - S_{39}: Missing Exception Flows Section$^{\diamond}$
  - S_{40}: Missing Alternate Flows Section$^{\diamond}$
  - S_{41}: Missing Preconditions Section$^{\diamond}$
  - S_{42}: Missing Postconditions Section$^{\diamond}$
  - S_{43}: Missing Description Section$^{\diamond}$
  - S_{44}: Missing Name Section$^{\diamond\bullet}$
- $C_{5,3}$: ⟨***Lack***, ***Flow***⟩
  - S_{45}: Premature Exceptional Cases
  - S_{46}: Premature Branch Condition
- $C_{5,4}$: ⟨***Lack***, ***Sentence***⟩
  - S_{47}: Exception Flow without Return$^{\diamond\bullet}$
  - S_{48}: Exception Flow with Conditions Undescribed$^{\diamond\bullet}$
  - S_{49}: Alternate Flow without Return$^{\diamond\bullet}$
  - S_{50}: Alternate Flow with Conditions Undescribed$^{\diamond\bullet}$
  - S_{51}: Incomplete System Behavior
  - S_{52}: Incomplete System Information
- $C_{5,5}$: ⟨***Lack***, ***Word***⟩
  - S_{53}: Missing Action Target
  - S_{54}: Missing Operation Procedure
  - S_{55}: Unknown Origin
- $C_{6,2}$: ⟨***Misplacement***, ***Section***⟩
  - S_{56}: Precondition in Basic Flow
  - S_{57}: Postcondition in Basic Flow
  - S_{58}: Exception Flow in Basic Flow
  - S_{59}: Alternate Flow in Basic Flow
- $C_{7,5}$: ⟨***Inconsistency***, ***Word***⟩
  - S_{60}: Synonym*

The smell types annotated with "•" in Table 2 can be considered as syntactic errors and/or semantic ones rather than symptoms of poor descriptions that possibly leads to problems. As well known, we have several dialects of use case descriptions, and the decision on syntactic and/or semantic errors greatly depends on which dialects of use case descriptions we adopt. As mentioned in Sect. 3, we have referred to [8] for use case descriptions, and using its meta-model, we have decided which smell types are syntactic errors and/or semantic ones. If we adopt the dialects different from [8], its results may be different.

Note that Table 2 lists up totally 60 smell types, and six smell types annotated with "*" in this table were not identified in this study and will be introduced later.

Table 3 Smell characteristics

Characteristic	Description
Ambiguity	The meaning cannot be determined uniquely, and its understanding requires effort.
Incorrectness	The content is incorrect or inconsistent.
Granularity	The content is too rough or too detailed.
Redundancy	There is an extra unnecessary part in the content.
Lack	There is a missing necessary part.
Misplacement	The content is located where it should not be located.
*Inconsistency**	Two of the contents are inconsistent.

Table 4 Smell scope

Scope	Description
Usecase	A problem is found in an entire use case description.
Section	A problem is found in a specific section such as "Actors" or "Description."
Flow	A problem is found in a specific flow or its subset representing a sequence of steps. A typical case is that two steps in it are inconsistent.
Sentence	A problem is found in a specific sentence in a description.
Word	A problem is found in a word in a description.

Table 5 Smell space

Scope / Characteristic	L_1 *Usecase*	L_2 *Section*	L_3 *Flow*	L_4 *Sentence*	L_5 *Word*
1. *Ambiguity*	$C_{1,1}$	$C_{1,2}$	$C_{1,3}$	$C_{1,4}$	$C_{1,5}$
2. *Incorrectness*	$C_{2,1}$	$C_{2,2}$	$C_{2,3}$	$C_{2,4}$	$C_{2,5}$
3. *Granularity*	$C_{3,1}$	$C_{3,2}$	$C_{3,3}$	$C_{3,4}$	$C_{3,5}$
4. *Redundancy*	$C_{4,1}$	$C_{4,2}$	$C_{4,3}$	$C_{4,4}$	$C_{4,5}$
5. *Lack*	$C_{5,1}$	$C_{5,2}$	$C_{5,3}$	$C_{5,4}$	$C_{5,5}$
6. *Misplacement*	$C_{6,1}$	$C_{6,2}$	$C_{6,3}$	$C_{6,4}$	$C_{6,5}$
7. *Inconsistency*	$C_{7,1}$	$C_{7,2}$	$C_{7,3}$	$C_{7,4}$	$C_{7,5}$

Name: Long Sentence
Characteristic: *Granularity*
Scope: *Sentence*
Symptom: A sentence is relatively long to the other ones in a section. The target sentence may contain too much information or may contain unnecessary information, which decreases the understandability of the sentence. It should be separated so as to be easier to understand.
How to Detect: Checks whether the number of characters in a sentence (length of a sentence) exceeds a threshold which is calculated by the distribution of the lengths of the sentences among the target use case description.

Fig. 3 Smell description of Long Sentence.

3.3 Classification Criteria and Smell Types

We found that the derived smell types could be categorized in two views. The first view is general reasons why the smells are problematic, which we call it *smell characteristic* hereafter. We identified six characteristics: *Ambiguity*, *Incorrectness*, *Granularity*, *Redundancy*, *Lack*, and *Misplacement*. The meaning of each characteristic is explained in Table 3. For example, 12 out of 54 smell types were related to the ambiguity of some specific contents in a use case description, which were classified as *Ambiguity*. Note that the smell characteristic of *Inconsistency*, which is annotated with "*" in Table 3, was not identified in this study and will be introduced later.

The second view is the *scope* of smells. We found that the scope of identified smell instances diversified in a wide variety from small-scoped ones such as word level to large-scoped ones such as an entire use case description. Therefore, we also classified the smell types based on their scopes: *Usecase*, *Section*, *Flow*, *Sentence*, and *Word* levels. Their meanings are explained in Table 4. The scope means not a syntactic level of a problem that a bad smell can cause, but a level having the bad smell. Let us consider the bad smell S_{35}: Repeating the Same Noun shown in Table 2. This smell means that a sentence S contains multiple occurrences of the same noun N, and the sentence S can include redundant information denoted by the noun N. Obviously, the problem is on the multiple occurrences of N themselves, and it could be considered as a problem in word level. However, in our classification of smell types, as the bad smell from the multiple occurrences of N spreads over the sentence S, we classify its scope as "Sentence".

As mentioned before, we have two orthogonal views: Characteristic and Scope to categorize bad smells. Considering a view as a coordinate axis, we can have a two-dimensional space called *smell space*. As shown in Table 5, we may write a category of bad smells with a two dimensional coordinate or vector like $C_{3,4} = \langle Granularity, Sentence \rangle$ where the values of Characteristic-coordinate = *Granularity* and Scope-coordinate = *Sentence*.

Figure 3 shows an example of the documentation of our bad smell catalog. It contains the following sections:

Name: The name of the bad smell. It should be a sufficiently descriptive and unique name.
Characteristic and Scope: The characteristic and the scope of the bad smell, which are one of the items listed in Table 3 and one in Table 4, respectively. These also specify the category of the smell, which is based on Table 5. The smell example shown in Fig. 3 belongs to the category of $\langle Granularity, Sentence \rangle$.
Symptom: Explanation on the smell, including surface features of the descriptions and the problems that the smell can cause. It provides intuitive reasons why these surface features could be problematic.
How to Detect: More concrete descriptions on the surface features of the descriptions to detect the bad smell by manual and/or machine. This section can be used as a checklist like [4], [5], etc. Some of the features can be automated to detect and we have implemented them as the smell detector.

The identified smell instances (problems) were distributed over different smell categories, i.e., the smell space. Table 6 shows how many instances were classified in the categories defined in the smell space. Although the smell instances in *Section* and those for *Lack* were identified at most, those for other scopes and characteristics were also

Table 6 Numbers of identified smell instances

Characteristic	*Usecase*	*Section*	*Flow*	*Sentence*	*Word*	Total
Ambiguity		23		12	25	60
Incorrectness		16	7	2		25
Granularity	3	2		29	3	38
Redundancy				3	1	4
Lack	1	50	19	25	13	108
Misplacement		13				13
Total	4	104	26	71	42	247

identified.

4. Automating Smell Detection

4.1 Deriving Metrics via GQM

Next, to implement an automated smell detector, metrics of a use case description are derived by applying Goal-Question-Metric (GQM) paradigm [7] to the identified smells mentioned in the last subsection. GQM is a very popular method and widely used to derive metrics in software engineering communities. GQM application was conducted according to the following steps:

1. Top two layers were used as *Goal* part, and the goals in these layers were systematically derived. The categories in the smell space, i.e., pairs of the scope and the characteristic of smells, were used for the root goals. The specific types of the smells were used for the sub goals of these root goals in the second layer.
2. As *Question* part, questions were derived as to which features should be examined to detect the instances of each smell in the sub goal layer.
3. Finally, for each question, metrics to be used to answer the questions were derived as leaves in *Metric* layer.

An excerpt example of the derivation of metrics by applying GQM is shown in Fig. 4.

1. As a root goal, we considered the category of ⟨*Granularity*, *Sentence*⟩ and derived the goal "Detecting smells of an inappropriate granularity in a sentence" as a root goal. Then, we considered Long Sentence and Short Sentence as smells related to an inappropriate granularity in a sentence, and derived two sub goals "Detecting Long Sentence" and "Detecting Short Sentence" at the second layer.
2. In order to detect Long Sentence, the main question is whether the length of a given sentence is regarded as within the average, and the question "Are the length of all the sentences within an average?" was derived.
3. To answer the question above, we need to measure the length, i.e., the number of characters, of a sentence to determine whether the sentence length is within an average range of the sentence lengths.

Metrics derived as leaves could be implemented as LOS (The length of a sentence). The same approach was applied to the other smells to derive metrics. There are two types of metrics to be derived: not only those that output

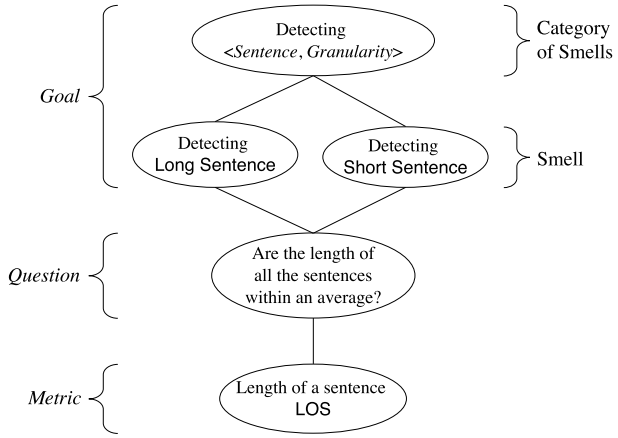


Fig. 4 Excerpt example of GQM application for Long Sentence and Short Sentence.

a numeric value such as LOS as explained above, but also those that determine whether the given content of a use case description satisfies a specific condition. We call the formers and the latters *numeric metrics* and *predicates*, respectively. Predicates can be implemented as metrics that output a boolean value.

Note that we used the quartile of the LOS values in each use case to decide threshold values for detecting Long Sentence and Short Sentence. It means that the threshold values are not constant ones but relatively changed according to the set of sentences in a use case. More concretely, in the case of Short Sentence, the threshold value $Threshold_{min}$ is defined as:

$$Threshold_{min} = 1^{st} - 1.5 \times (3^{rd} - 1^{st})$$

where 1^{st} and 3^{rd} are 25% and 75% percentiles, respectively. If LOS is less than $Threshold_{min}$, the sentence is decided as Short Sentence.

The reason why we adopt the category of smells and smell types as a root and a second one respectively is that in the same category of smells, we may share with the idea of questions and metrics. In fact, as shown in Fig. 4, the question "Are the length of all the sentences with in an average?" is a common child of the two goals "Detecting Long Sentence" and "Detecting Short Sentence". Sharing with the idea of questions and metrics can assist us in deriving metrics as guidelines.

By applying GQM, machine-measurable metrics were derived for 22 smell types. In deriving machine-measurable metrics, we limited the analysis scope within the syntactic analysis of natural language descriptions and structural analyses of a use case description. Of the smells shown in Table 2, those annotated with "⋄" are detectable using the derived smells. Tables 7 and 8 list the derived numeric metrics and predicates, and the columns "Smells" in the tables show the smells that metrics and predicates are used to detect. Note that, similarly to Table 2, numeric metrics annotated with "*" shown in Table 7 were not derived from the analysis in this section, and they were derived during the ex-

Table 7 Numeric metrics to detect smells

Name	Description	Smells
NOP	The number of pronons	S_7
NOEFC*	The number of exception flows with the same condition	S_{23}
NOAFC*	The number of alternate flows with the same condition	S_{24}
LOS	The length of a sentence	S_{25}, S_{26}
NOV	The number of verbs	S_{27}
NOM	The number of modifiers	S_{28}, S_{29}
NON(n)	The number of the noun word n	S_9, S_{35}

Table 8 Predicates to detect bad smells

Name	Description	Smells
BasicFlowNumbered?	Are all the steps in the given flow numbered?	S_1
ExceptionFlowsNumbered?	Are all the steps in the exception flows numbered?	S_1
AlternateFlowsNumbered?	Are all the steps in the alternate flows numbered?	S_1
BasicFlowOrdered?	Are all the steps in the basic flow well ordered?	S_1
ExceptionFlowsOrdered?	Are all the steps in the exception flows well ordered?	S_1
AlternateFlowsOrdered?	Are all the steps in the alternate flows well ordered?	S_1
BasicFlowStartWith1?	Does the basic flow start with 1?	S_1
ExceptionFlowsStartWith1?	Is the index of the first step of the exception flows 1?	S_1
AlternateFlowsStartWith1?	Is the index of the first step of the alternate flows 1?	S_1
ExceptionFlowsOriginDescribed?	Is the origin of the exception flows described?	S_2
AlternateFlowsOriginDescribed?	Is the origin of the alternate flows described?	S_3
ActorSectionExist?	Does the Actors section exist?	S_{38}
ExceptionFlowsSectionExist?	Does the Exception Flows section exist?	S_{39}
AlternateFlowsSectionExist?	Does the Alternate Flows section exist?	S_{40}
PreconditionsSectionExist?	Does the Precondition section exist?	S_{41}
PostconditionsSectionExist?	Does the Postcondition section exist?	S_{42}
OverviewSectionExist?	Does the Overview section exist?	S_{43}
NameSectionExist?	Does the Name section exist?	S_{44}
ExceptionFlowsReturnExist?	Is it described where the exception flows return to?	S_{47}
AlternateFlowsReturnExist?	Is it described where the alternate flows return to?	S_{49}
ExceptionFlowsConditionExist?	Are the conditions when exception flows are executed described?	S_{48}
AlternateFlowsConditionExist?	Are the conditions when alternate flows are executed described?	S_{50}

periment. The details will be explained in the next section.

Instead of these 22 smell types, we failed to derive machine-measurable metrics for the other 32 smell types. A typical reason why machine-measurable metrics derivation was failed for such smell types is that they were related to the semantic concepts and/or domain knowledge. For example, consider trying to derive metrics for Multiple Actors in a Role, which indicates that two different actors actually have the same role. For example, an actor named "Family" in the context of medical care may have the role of liaison between a patient and a doctor, which causes a situation that two actors having different names have the same semantic role. However, to detect smells of this type, we need to extract the semantic role of actors, which is unable only with syntactic and structural analyses.

4.2 Smell Detector

To evaluate the metrics that could be automated, we have implemented a prototype of the automated smell detector with object-oriented script language Ruby[†]. The reasons why we used the script language are that

1. we can add new metrics and change the metrics easily and flexibly, and
2. we can integrate the functions of the smell detector to the other use case supporting tools.

For implementing the detector, we used MeCab[††] as a morphological analyzer and NEologd[†††] [9] as a dictionary to analyze Japanese sentences in use case descriptions. The outputs of the detector are the information on detected bad smells including locations, used metrics, etc. The detection rules of the smell types annotated with "◇" shown in the list in Table 2 have been implemented in the detector.

Figure 5 shows the snapshot of the output of our detector for the use case "Withdraw money with ATM", which is shown in the left part of Fig. 6. Our detector is only for calculating our metrics and indicating bad smells in use case descriptions and for evaluating if our metrics can capture the occurrences of bad smells. Thus, it has no graphical user interface yet. It takes as inputs the texts of use case descriptions that can be processed by MeCab, and it outputs detection results in a textual form.

The output follows the Ruby object literal format. A string parenthesized with "`{`" and "`}`" denotes an occurrence of detected bad smells, and it normally consists of four columns. The first column headed with "`item_name`", the second with "`metric`", the third with "`line`", and the

[†]https://www.ruby-lang.org/

[††]https://taku910.github.io/mecab/

[†††]https://github.com/neologd/mecab-ipadic-neologd

```
[{:item_name=>"pre_conditions", :metric=>:do_not_describe_subject_as_the_word_actor, :line=>1, :word=>"アクター"},
 {:item_name=>"post_conditions", :metric=>:do_not_describe_subject_as_the_word_actor, :line=>1, :word=>"アクター"},
 {:item_name=>"main", :metric=>:do_not_describe_subject_as_the_word_actor, :line=>1, :word=>"アクター"},
 {:item_name=>"main", :metric=>:do_not_describe_subject_as_the_word_actor, :line=>3, :word=>"アクター"},
 {:item_name=>"main", :metric=>:do_not_describe_subject_as_the_word_actor, :line=>5, :word=>"アクター"},
 {:item_name=>"main", :metric=>:do_not_describe_subject_as_the_word_actor, :line=>7, :word=>"アクター"},
 {:item_name=>"main", :metric=>:do_not_use_pronoun, :line=>9, :word=>"それ"},
 {:item_name=>"main", :metric=>:do_not_describe_subject_as_the_word_actor, :line=>9, :word=>"アクター"},
 {:item_name=>"alternate", :metric=>:do_not_describe_subject_as_the_word_actor, :line=>"1-0", :word=>"アクター"}]
[{:item_name=>"main", :metric=>:count_of_verb, :line=>6,
  :sentence=>"システムは、口座の残高を検査し、引き出し金額を残高から引いて、引き出し金額をお金の取り出し口に出す。"}]
[{:metric=>:duplicate_alternate_flows, :use_case_name=>"お金を引き出す"},
 {:metric=>:missing_actor_column, :use_case_name=>"お金を引き出す"}]
```

Actor

it

System checks the balance of the account, removes the withdrawn amount from the balance and puts the withdrawn amount to the output port of the ATM.

Withdraw money with ATM

Fig. 5 Snapshot of the tool output.

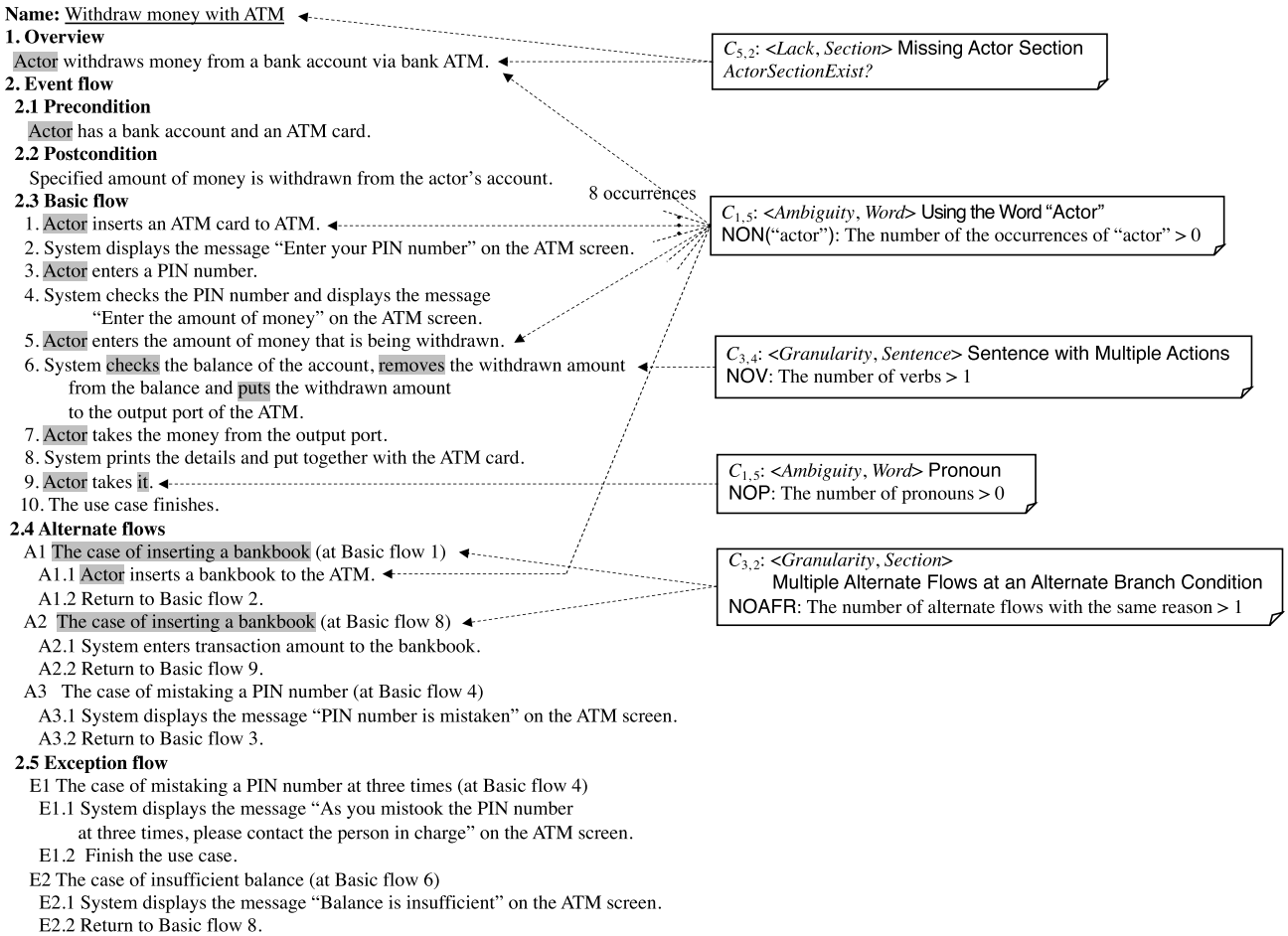


Fig. 6 Example result of the smell detection.

fourth with "`word`", "`sentence`", or "`flow`" represent the location of the occurrence in the use case description, the used metrics to detect it, the line number of it in the location, and its string data, respectively. In the example of the first occurrence of the detected bad smell, it appeared in the section "Precondition", the metrics NON("actor") (more comprehensively shown as *do not describe subject as the word* "*actor*") was used, it appeared in the first line of the precondition section and the word "Actor" in the line was caused the bad smell. Since our metrics work for Japanese, our tool picked up a Japanese word, a Japanese sentence, and a set of Japanese sentences (a flow) as a bad smell. In Fig. 5, the readers can find English translation of the detected underlined Japanese words and sentences.

Figure 6 shows the example of a use case "Withdraw money with ATM" and its detection result. This figure is not a screenshot of our detector, but for showing the detection result comprehensively. We put the detection result of the tool on the use case description together with short texts of clarifying detected bad smells. The detected bad smells are listed on the right side of the figure. The first bad smell is missing actor sections belonging to the category $C_{5,2}$, whose characteristic and scope are *Lack* and *Section*, respectively. To detect this bad smell, the predicate *ActorSectionExist?* in

Table 8 returned the value False because our detector found that there were no sections on the specification of actors in the use case description. In Fig. 6, the bad smell S_9 (Using the Word "Actor") of category $C_{1,5}$ appeared in the eight sentences, whose subject is the word "actor". If this use case was related to multiple actors, the word "actor" was ambiguous because we could not decide which actors the word "actor" denoted. Our detector decided that the eight occurrences of the word "actor" caused a bad smell Using the Word "Actor" using the metrics NON("actor"), i.e., the number of the occurrences of "actor", in Table 7. The sixth sentence in "2.3 Basic Flows" section was decided to have the bad smell Sentence with Multiple Actions because it included three verbs "checks", "removes", and "puts" and thus was a compound sentence. The usage of pronouns caused ambiguity, and our detector detected the pronoun "it" in the ninth sentence of Basic flows. The last category of the detected bad smell was Multiple Alternate Flows at an Alternate Branch Condition. In "2.4 Alternate flows" section, the branch conditions to alternate actions of A1 and A2 were the same. It means that A1 and A2 should be executed under an alternate condition, and for easiness to read, they should be modularized together under the condition.

5. Analytic and Experimental Evaluation

In this section, we discuss the evaluation of our approach: the developed catalog and the automated smell detector.

5.1 Research Questions

First of all, we set up the following research questions (RQs) to validate our approach.

RQ_1: Can our catalog cover various bad smells?
RQ_2: Can our automated detector indicate all of the occurrences of bad smells correctly?

RQ_1 is concerned with the coverage of our catalog. We validate it from two views: an analytic view and an experimental one. In the former one, we collect the existing popular checklists of use case descriptions and compared ours with them. As for the latter one, we ask the study participants to find bad smells from some examples of use case descriptions and assess if their results would be included in our catalog or not. Thus, RQ_1 can be refined into the following two:

$RQ_{1\text{–}1}$: Have our catalog included the bad smells that study participants have found in use case description examples?
$RQ_{1\text{–}2}$: Can our catalog cover some existing popular checklists?

To respond RQ_2, we adopt an experimental comparative approach where we compare the correct set of bad smells with the results that our detector indicates. We reuse as a correct set the results that the study participants produced in RQ_1.

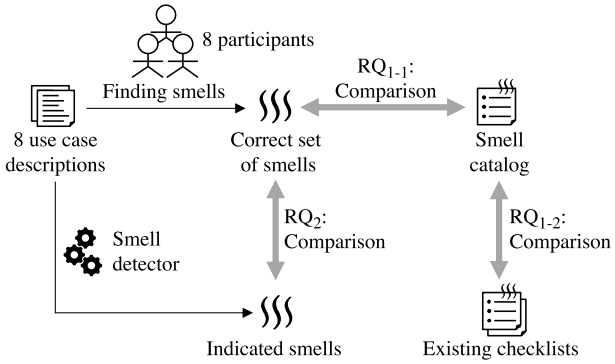


Fig. 7 Overview of the evaluation process.

The overview of our evaluation process is shown in Fig. 7 and its details will be mentioned in the next subsection.

5.2 Evaluation Procedure

As shown in Fig. 7, we had eight examples of use case descriptions, which were different from the previous 30 examples that we used to develop the catalog and the metrics, and eight study participants who had experienced in use case modeling. The study participants consisted of two undergraduate and five graduate students, and one faculty in the department of computer science majoring in software quality and/or requirements engineering, who were gathered via the authors' network. They have half to 15 years of experience in use case modeling. Our participants were asked to find in the examples the poor descriptions that might cause any of various problems in some aspects, e.g., understandability, realizability, implementability, consistency, etc. They were also required to clarify the reasons why they judged their findings as poor descriptions. We first gathered all the participants at the same time in a single room and conducted a face-to-face meeting. During the meeting, we explained the overview of the experiment, distributed printed sheets of use case descriptions, provided 30 min to find the examples, and responded to questions from the participants. After the meeting, participants were allowed to complete to find examples separately without any time limitation. The examples were specified in the printed sheets as their location and reason. We analyzed the reasons for the found poor descriptions and tried to classify them into bad smells of our catalog. We regarded the classification results as a correct set of bad smells included in the examples. The correct set is used for responding $RQ_{1\text{–}1}$ and RQ_2.

These eight examples are the remainder of the 38 use case descriptions as mentioned in Sect. 3.1. It is difficult to create a complete and real correct set of bad smells because the judgment of bad smells or not is subjective to human analysts. Thus, we took the poor descriptions that at least one of the eight study participants considered as poor descriptions so that we could capture the candidates of bad smell instances as many as possible. One of the authors confirmed

all the obtained smell instances. During the confirmation, only one of the participants' results were excluded because that was based on a misreading of the description.

As for RQ_{1-1}, some of the bad smells in the correct set could not be classified into our catalog, and we consider these bad smells as the new bad smells that should be added to the catalog. We counted the number of the new bad smells to respond RQ_{1-1}.

To respond RQ_{1-2}, we selected three popular checklists made by

1. Phalp *et al.* [4],
2. Törner *et al.* [5], and
3. Anda and Sjøberg [10],

and tried to compose a mapping between our catalog and their check items. The mapping allows us to analyze if the bad smells of our catalog cover their check items or not. More precisely, we focus on the bad smells of our catalog that cannot be mapped to any of their check items (we say, a bad smell is not included in a checklist) and on the check items that cannot be mapped to any of our bad smells (a checklist item is not included in our catalog). Since the granularity of our catalog may be different from that of their check items, the mapping cannot necessarily be one-to-one, but one-to-many or many-to-many.

We applied our smell detector to the eight examples and compared the results with the correct set. More concretely, we listed up

1. the bad smell instances that were included in the correct set and also indicated by the detector (true positives: *TP*),
2. the bad smell instances included in the correct set but not indicated by the detector (false negatives: *FN*), and
3. the descriptions not included in the correct set but indicated by the detector (false positives: *FP*).

To respond RQ_2, we calculate a precision ratio and recall one as follows:

$$Precision = \frac{|TP|}{|TP| + |FP|}, \qquad Recall = \frac{|TP|}{|TP| + |FN|}.$$

5.3 Results

5.3.1 Response to RQ_{1-1}: Comparison with the Correct Set

The experiment that our eight study participants conducted made us new six bad smells that were not included in the first version of our catalog, the bad smells annotated with "*" shown in Table 2. As a result, we have two versions of bad smell catalog; the first version has 54 bad smells, and the improved version does 60 bad smells, and furthermore, we have developed the metrics to indicate these six newly added bad smells in the same way, i.e., using GQM approach. As a result, we could finally get 30 metrics, which respectively consist of 8 numeric metrics and 22 predicates shown in Tables 7 and 8, for the improved version of our catalog, by adding new two numeric metrics, which are annotated with "*" in Table 7, to detect two of the newly added bad smells. In the later subsections of responding RQ_{1-2} and RQ_2, we compare these two versions of catalog with the existing checklists (RQ_{1-2}). We also use two versions of metrics to respond RQ_2.

5.3.2 Response to RQ_{1-2}: Comparison with Existing Checklists

Table 9 shows the comparative results with the existing three checklists. The checklists of Phalp *et al.* [4], Törner *et al.* [5], and Anda and Sjøberg [10] have 13, 12, and 27 bad smell types, respectively. The cell with the symbol ✓indicates that the smell type defined in our catalog, shown in rows, can be associated with the smell type in the checklist, shown in columns.

Table 10 shows the summary of the comparison. Some cells in the table include two numerals together with the symbol ⇒. The numeral in the left hand side of ⇒ stands for the value for the first version of our catalog, and the right does for its improved version. For example, Phalp *et al.* [4] has two bad smell types that are not included in the first version of our catalog, and they decrease to one for the improved version.

On the contrary, at first our catalog had four bad smells that are not included in any of these checklists, and it could be improved so that it has six bad smell types not included in any of these existing checklists: Omitted Attribute, Multiple Exception Flows at an Exceptional Branch Condition, Multiple Alternate Flows at an Alternate Branch Condition, Multiple Roles of an Actor, Multiple Actors of a Role, and Non-Standalone Use Case.

On the other hand, we did not have six bad smells (one smell from Phalp *et al.*, one smell from Törner *et al.*, four smells from Anda and Sjøberg) in total as follows.

1. Consideration of Alternatives: Viable (Phalp *et al.*, #12 in Table 9): It is a smell where alternate flows cannot be really executed. Its category in our catalog depends on the reasons of the impossibility of executing the alternative flow. In the case where the reasons to branch to the alternative flow are unexplained at all and so it is impossible to branch to the flow, the category of this smell can be S_{45}, S_{48}, or S_{50}. In the case that the reasons are ambiguous or inconsistent, we can add this smell as new one to the category $\langle Ambiguity, Flow\rangle$ or $\langle Inconsistency, Flow\rangle$, respectively. If there are any problems in the flow executed after the branch, this smell can be S_{18}, S_{23}, S_{24}, S_{34}, S_{47}, or S_{49} according to the causes of the problems.
2. Use Case Decomposition (Törner *et al.*, #7 in Table 9): In this smell, several activity flows should be extracted from the use case and moved into one or more new use cases. Thus, we can add it as a new one of the category $\langle Granularity, Usecase\rangle$.

Table 9 Comparison among smell catalogs

	Phalp *et al.* [4]												Törner *et al.* [5]												Anda and Sjøberg [10]																															
	01	02	03	04	05	06	07	08	09	10	11	12	13	01	02	03	04	05	06	07	08	09	10	11	12	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
S_1: Unordered Flow			✓						✓				✓					✓					✓																																		
S_2: Origin-Free Exception Flow																						✓																																		
S_3: Origin-Free Alternative Flow																						✓																																		
S_4: Unclear Feasibility																				✓																																			
S_5: Origin-Free Operation Result																				✓							✓																												
S_6: Sentence Interpretable as Multiple Meanings																																										✓		✓	✓	✓							
S_7: Pronoun										✓													✓																				✓		✓	✓	✓							
S_8: Omitted Word										✓																																		✓	✓	✓							
S_9: Using the Word “Actor”										✓																																												
S_{10}: Undeclared Main Actor										✓													✓			✓																												
S_{11}: Different Concepts by Same Word										✓													✓																																
S_{12}: Omitted Attribute																																																							
S_{13}: Name Does Not Explain Content	✓	✓			✓																✓											✓												✓									
S_{14}: Too Weak or Strong Condition	✓	✓			✓										✓					✓					✓																			✓									✓
S_{15}: Much or Less Actors	✓	✓												✓																																								
S_{16}: Flow Does Not Meet Precondition	✓				✓																			✓										✓	✓	✓																	
S_{17}: Postcondition Not Satisfied	✓				✓										✓									✓										✓	✓	✓			✓													
S_{18}: Contradicted Sentences					✓																			✓										✓	✓																	
S_{19}: Behavior Ignores Condition					✓																			✓										✓	✓																	
S_{20}: Multiple Situations																			✓																				✓														
S_{21}: Multiple Roles of an Actor																																																					
S_{22}: Multiple Actors of a Role																																																					
S_{23}: Multiple Exception Flows at an Exception Branch Condition																																																				
S_{24}: Multiple Alternate Flows at an Alternate Branch Condition																																																				
S_{25}: Long Sentence							✓												✓				✓																												✓		
S_{26}: Short Sentence							✓																✓																					✓								
S_{27}: Sentence with Multiple Actions							✓												✓				✓																												✓		
S_{28}: Relatively Over-Qualified Sentence							✓																✓																												✓		
S_{29}: Relatively Under-Qualified Sentence							✓																✓																					✓								
S_{30}: Less Qualified Word in Following Sentences						✓																	✓																														
S_{31}: Less Qualified Word in Above Sentences						✓																																														
S_{32}: Multiple Flows with the Same Role		✓																✓																																				
S_{33}: Flow Unrelated to Postcondition		✓		✓												✓		✓																						✓	✓									✓		
S_{34}: Conditional Flow																		✓																						✓			✓									
S_{35}: Repeating the Same Noun										✓												✓																																
S_{36}: Over-Qualified Word																							✓																														
S_{37}: Non-Standalone Use Case																																																				
S_{38}: Missing Actor Section														✓												✓																										
S_{39}: Missing Exception Flows Section										✓				✓								✓						✓																								
S_{40}: Missing Alternate Flows Section										✓				✓								✓						✓																								
S_{41}: Missing Preconditions Section														✓						✓																																
S_{42}: Missing Postconditions Section														✓						✓																																
S_{43}: Missing Description Section														✓													✓																								
S_{44}: Missing Name Section														✓																														✓							
S_{45}: Premature Exceptional Cases										✓												✓						✓																								
S_{46}: Premature Branch Condition										✓												✓						✓																								
S_{47}: Exception Flow without Return																✓																													✓							
S_{48}: Exception Flow with Conditions Undescribed																						✓																								✓							
S_{49}: Alternative Flow without Return																✓																																				
S_{50}: Alternative Flow with Conditions Undescribed																						✓																								✓							
S_{51}: Incomplete System Behavior	✓														✓												✓																								
S_{52}: Incomplete System Information	✓														✓												✓																								
S_{53}: Missing Action Target	✓																																																			
S_{54}: Missing Operation Procedure	✓																																																		
S_{55}: Unknown Origin	✓																																																		
S_{56}: Precondition in Basic Flow																			✓																																		
S_{57}: Postcondition in Basic Flow																			✓																																		
S_{58}: Exception Flow in Basic Flow								✓														✓																																
S_{59}: Alternate Flow in Basic Flow								✓														✓																																
S_{60}: Synonym						✓																	✓																														

3. Incorrect description of actors or wrong connection between actor and use case (Anda and Sjøberg, #7 in Table 9): This smell can be considered as a new smell of the category ⟨*Incorrectness*, *Section*⟩ related to actors.
4. Inconsistencies between diagram and descriptions, inconsistent terminology, inconsistencies between use cases, or different level of granularity (Anda and Sjøberg, #16 in Table 9): From the view of our classification of bad smells, the category of this smell can be considered as a mixture of several categories. As for the smell Inconsistencies between diagram and descriptions, we did not consider use case diagrams, and so it is outside of the current version of our catalog. The smell Inconsistent terminology can belong to the category ⟨*Inconsistency*, *Word*⟩. Depending on the cause of Inconsistency, it can also be the smell S_{60}. The smell Inconsistencies between use cases, or different level of granularity can be categorized to ⟨*Inconsistency*, *Usecase*⟩ or ⟨*Granularity*, *Usecase*⟩, respectively.
5. Actors that do not derive value from/provide value to the system (Anda and Sjøberg, #23 in Table 9): This smell can be considered as the case where wrong or unnecessary actors are specified, so we can put it as a new smell in the category ⟨*Redundancy*, *Section*⟩.
6. Use cases with functionality outside the scope of the system or use cases that duplicate functionality (Anda and Sjøberg, #24 in Table 9): Similarly, this smell can be considered as the case where unnecessary use cases are specified, so we can put it as a new smell in the category ⟨*Redundancy*, *Usecase*⟩.

To sum up, as our classification criteria are different from the other checklists, all of the six smells cannot be added as new smells to our categories one by one. In particular, some of the six smells can be new smells that we did not find in our experiments. The others are compound ones in the sense that they can be put in more than one category of ours, and they can be divided into one category by their causes. One smell deals with use case diagrams, but the use case diagrams are out of scope in the current version of our catalog.

Table 10 Comparison with existing checklists

Checklist	# smell types	# smell types not included in our catalog
Phalp *et al.* [4]	13	2 ⇒ 1
Törner *et al.* [5]	12	2 ⇒ 1
Anda and Sjøberg [10]	27	6 ⇒ 4
		# smell types not included in [4], [5] or [10]
Our catalog	54 ⇒ 60	4 ⇒ 6

5.3.3 Response to RQ_2: Comparison the Results of Tool Application with the Correct Set

Table 11 shows the results for each bad smell category included in the eight examples. It also includes the number of indications by the smell detector (# detector indicated) and the number of the bad smell instances actually included in the examples (# smell instances in the correct set), which our study participants indicated. Totally, we found that the examples actually included 53 bad smell instances, and our tool could indicate 88 bad smell instances. 52 of the 88 bad smell instances were in the correct set, i.e., *TP*. For example for the bad smell category ⟨*Ambiguity*, *Section*⟩, we got the 12 bad smell instances that the detector indicated, and they included all of the ten correct bad smell instances.

Some of the cells in the table include two numerals with the symbol ⇒. As mentioned in Sect. 5.1, through constructing the correct set we found new additional six bad smell types and added them to the catalog, and accordingly, we added several metrics so as to indicate these new bad smells. The numerals in the left hand side of ⇒ stand for the results obtained by the detector before adding the new metrics and the right represents the result by the detector improved by adding the metrics. In total, our improved detector indicated 89 bad smell instances, 53 of the 89 were correct.

High recall values could be partially satisfied to RQ_2, while lower precision values could not be sufficient. We found two major reasons for the lower precision values. The first one was the detection of ambiguous words: ⟨*Ambiguity*, *Word*⟩. We consider the word "actor" is ambiguous because some use cases had multiple actors. However, in the case of the use cases that have only one actor, our study participants did not consider that the word "actor" appearing in the sentences was ambiguous because they could identify correctly an object that the word "actor" denotes. The second reason was related to the bad smell category ⟨*Granularity*, *Sentence*⟩. We used the relative length of sentences as one of the metrics to detect smells related to the

Table 11 Precision and recall values for bad smell detection

Category	Precision	Recall	# detector indicated	# smell instances in the correct set
⟨*Ambiguity*, *Section*⟩	0.833	1.000	12	10
⟨*Ambiguity*, *Word*⟩	0.522	1.000	23	12
⟨*Granularity*, *Flow*⟩	N/A ⇒ 1.000	0.000 ⇒ 1.000	0 ⇒ 1	1
⟨*Granularity*, *Sentence*⟩	0.385	1.000	26	10
⟨*Redundancy*, *Sentence*⟩	N/A	N/A	0	0
⟨*Lack*, *Section*⟩	0.696	1.000	23	16
⟨*Lack*, *Sentence*⟩	1.000	1.000	4	4
Total	0.591 ⇒ 0.596	0.981 ⇒ 1.000	88 ⇒ 89	53

category ⟨*Granularity*, *Sentence*⟩. Some sentences included illustrative phrases, and as a result, they came to be relatively long to the other sentences. However, our study participants judged that these sentences including illustrative phrases were easier to understand, even if they were long. This is the reason why our detector indicated bad smells incorrectly.

5.4 Threats to Validity

5.4.1 External Validity

External validity is related to the generality of the obtained conclusions. In this experiment, although we used only eight use case descriptions, their problem domains were different, and we believe that they covered varieties of the problem domain in a certain degree.

To evaluate the quality of our produced catalog, we compared it with the other existing catalogs. However, to compose the catalog, we used a limited set of use case descriptions (30+8). In fact, during the evaluation process, we used different eight use case descriptions, which our study participants analyzed, and it led to adding new six smells. It means that the entries of our catalog can be newly found and added whenever we focus on new use case descriptions. We do not think that our catalog would be complete or be a saturated catalog. However, our two orthogonal views, i.e., Characteristic and Scope, may be helpful to find new smells and to categorize them in the catalog, in the sense that human analysts can look for new smells with these two orthogonal views clues.

Also, the collected sample use case descriptions may not be enough regarding the generality. Although we have tried to increase the variety of samples by including those of multiple domains, we cannot guarantee that the samples are representative of the use case descriptions in general.

5.4.2 Internal Validity

We should explore the factors affecting the obtained results other than those of our approach. One of the possibilities of these factors is bias at constructing the correct set of bad smell instances by our eight study participants. However, their skills were sufficient and had experience in writing use case descriptions, and so we believe that the quality of the correct set was sufficient. In addition, they performed their tasks independently and there are no effects among our study participants.

The second factor was the possibility of developing metrics arbitrarily. However, we separated the examples of use case descriptions into two disjoint sets; one (30 examples) was for constructing the metrics and another (8 examples) for evaluating our detector.

The third factor was categorizing bad smells by one person when the bad smell catalog was developed. It leads to the possibility to wrong categorization. Thus, the validation of the catalog done by multiple persons is necessary as one of the future work.

The fourth one is the problem domains whose use case descriptions we used. We do not necessarily think that we could cover all problem domains. Use case descriptions of a certain problem domain allows us to get the new entries of the catalog and the different evaluation results. Clarifying a role of problem domains is also one of the future work.

The fifth one is that we did not have the agreement process of the correct set of smell instances by the study participants, and we used the union of the results of the participants. Although one of the authors confirmed all of the results except for one were valid, one can disagree with smell instances produced by another. Preparing smell instance oracle of high quality is also regarded as one of the future work.

6. Related Work

We focus on the studies to detect poor use case descriptions. As mentioned in Sect. 5.3.2, we could find three popular checklists. Phalp *et al.* [4] developed the checklist called 7C's to measure the quality of use case descriptions. It was based on an investigation about what quality was preferable for use case descriptions from the view of the understandability of natural language sentences. Törner *et al.* [5] developed the 12 criteria and provided the questionnaires written in the natural language to decide if descriptions are bad smells or not. These 12 criteria could be used to detect seven bad smells that Törner *et al.* listed up. Anda and Sjøberg [10] proposed the classification technique of flaws in use case models based on their locations and characteristics, and provided a checklist constructed from this classification. The checklist consists of questionnaires written in natural language related to locations and characteristics. The differences of these checklists to ours were discussed in Sect. 5.3.2 from the view of the coverage of bad smells. In addition, all of them are written in natural language, and the sufficient skills of their users are required.

Some researchers have adopted natural language techniques to analyze ambiguity and vagueness of natural language sentences as requirements descriptions. Yang *et al.* discussed nocuous ambiguity such as the scope of conjunctives "and" and "or", and automated to detect its occurrences [11], [12]. They applied their approach to usual natural language sentences, not use case descriptions. Use case descriptions are rather short sentences and may be incomplete ones. Liu *et al.* [13] tried to detect defects in use case documents automatically. In this approach, a use case description is transformed into an activity diagram based on dependency parsing techniques, and defect detection is applied to the obtained activity diagram. Fantechi *et al.* [14] summarized the application of linguistic techniques to use case analysis. Although their approach is complementary to ours, this line of approaches based on natural language processing is effective because it can extract a partial meaning of a use case description and can be utilized to increase the detection coverage of our smell catalog. Text2Test [15]

is an integrated environment for authoring use cases with a model-based checking is available. It is beneficial to develop a similar environment to realize Just-in-Time smell detection.

Refactorings, transformations that solve bad smells, have been used mainly in source code level [6], but the framework and terminology are not limited to the area related to source code. The framework, i.e., detecting quality problems of software artifacts as smells and fixing them by refactoring to improve the quality with preserving their core aspects, is straightforward and is applicable to software artifacts in the scope of requirements engineering, such as use case models [16], [17], feature models [18], goal models [19], and natural language documents [20], [21]. The goal of our approach can be classified as the same category; it regards the detection of problematic portions in use case documents as bad smells. The techniques on refactoring use case models [16], [17] focus on their structure improvement as model transformation, but not on the detection of bad smells. Their approaches are complementary to ours.

7. Conclusion

This paper discussed the automated technique to detect the occurrences of poor parts in use case descriptions. After clarifying bad smells (symptoms) of poor descriptions from the two views: characteristics and scope, we defined a catalog of bad smells for use case descriptions. Furthermore, we applied the GQM approach to the bad smells and developed metrics to automate the detection of bad smells. Finally, we could get the catalog of 60 bad smells and an automated smell detector to indicate 24 bad smells of them. Our analytic and experimental results showed that our direction is promising and some improvement is necessary. As a result, we could automate to detect the bad smells of syntax-related issues only. By capturing the catalog with a bird's eye view, as the benefit of the catalog, we can classify on the catalog the bad smells that could be automated, and the others, which could not be done, are semantic-related issue. Developing more sophisticated techniques for processing semantics of natural language is necessary to automate to detect them, and it is one of the future work. However, we think that our catalog can be used as a check list for human analysts to detect in industry. The categorization of bad smells on the catalog helped us to develop reusable metrics and predicates for detecting the bad smells of the same category. Same or similar metrics and predicates could be used to detect the bad smells of the same category and it allowed us to develop the tool without redundant efforts.

Future work can be summarized as follows:

- More experimental analyses on use case models of wide varieties of problem domains and in a practical level.
- Smell catalog: Collecting more use case descriptions with wide variety of problems domains and with practical levels to improve our catalogs and metrics. We used a limited set of use case descriptions, and it caused to be the possibility of the catalogs which could not cover all of bad smells yet. Accumulating various types of use case descriptions is useful to make the catalog more comprehensive. In addition, we should investigate whether types of our bad smells are specific to problems domains or not and clarify the roles of the problems domains to the bad smells.
- Threshold values of metrics: As for our current metrics, it is significant how to decide their suitable threshold values. In the example of Short Sentence and Long Sentence, we adopted relative threshold values on sentences in a use case, based on a quartile method. More experimental analysis may be helpful to decide if the threshold values calculated by a quartile method are suitable. As for the other threshold values we used, we need more experimental analysis to validate their suitability.
- Redoing categorization of bad smells by multiple persons and reviewing it to improve the quality of the catalog. As discussed in Sect. 5.4, when we developed the catalog, only one person categorized the obtained bad smells. To improve the objectivity of the categorization, we should have an involvement of multiple persons to categorize the bad smells again and to review the catalog.
- Developing a bad smell catalog for use case diagrams and an automated tool to detect the bad smells. Combining them with the results of this work can be considered.
- Enhancing metrics, in particular for detecting bad smells related to semantics. Our current metrics are for syntactic or surface features of use case descriptions. To get a more powerful detector, we need a deeper analysis of natural language sentences including semantic processing. In addition, since a use case model includes use case diagrams, the information that they have may be useful.
- More elaborated natural language processing: We used a simple morphological analysis only for natural language processing by MeCab, and it led us not to detect some occurrences of certain kinds of bad smells. In the example of the sentences including multiple occurrences of the nouns denoting actions, e.g., the noun whose concatenation to the word "Suru" makes a verb, our approach cannot detect the bad smell Sentence with Multiple Actions. By additionally using a dependency analyzer such as CaboCha[†], we can detect Sentence with Multiple Actions in this type of sentence.
- Technique to support the improvement of detected bad smells, so-called *refactoring* of use case models.

Acknowledgments

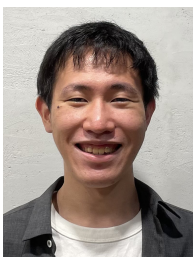
This work was partly supported by JSPS Grants-in-Aid for

[†]https://taku910.github.io/cabocha/

Scientific Research Nos. JP21K11823, JP18K11237, and JP18K11238.

References

[1] Y. Seki, S. Hayashi, and M. Saeki, "Detecting bad smells in use case descriptions," Proceedings of the 27th IEEE International Requirements Engineering Conference (RE'19), pp.98–108, 2019.
[2] S. Geri and W. Jason, Applying Use Cases: A Practical Guide, Addison Wesley Longman, 1998.
[3] M. El-Attar and J. Miller, "Improving the quality of use case models using antipatterns," Software and Systems Modeling, vol.9, no.2, pp.141–160, 2010.
[4] K.T. Phalp, J. Vincent, and K. Cox, "Assessing the quality of use case descriptions," Software Quality Journal, vol.15, no.1, pp.69–97, 2007.
[5] F. Tørner, M. Ivarsson, F. Pettersson, and P. Öhman, "Defects in automotive use cases," Proceedings of the 5th ACM/IEEE International Symposium on Empirical Software Engineering (ISESE 2006), pp.115–123, 2006.
[6] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison Wesley, 1999.
[7] V. Basili, C. Caldiera, and D. Rombach, "Goal, question, metric paradigm," Encyclopedia of Software Engineering, vol.1, pp.528–532, 1994.
[8] M. Misbhauddin and M. Alshayeb, "Extending the UML use case metamodel with behavioral information to facilitate model analysis and interchange," Software and Systems Modeling, vol.14, no.2, pp.813–838, 2015.
[9] T. Sato, T. Hashimoto, and M. Okumura, "Implementation of a word segmentation dictionary called mecab-ipadic-neologd and study on how to use it effectively for information retrieval (in Japanese)," Proceedings of the 23th Annual Meeting of the Association for Natural Language Processing (NLP 2017), 2017. NLP2017–B6–1.
[10] B. Anda and D.I.K. Sjøberg, "Towards an inspection technique for use case models," Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), pp.127–134, 2002.
[11] H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Extending nocuous ambiguity analysis for anaphora in natural language requirements," Proceedings of the 18th IEEE International Requirements Engineering Conference (RE'10), pp.25–34, 2010.
[12] H. Yang, A. Willis, A. De Roeck, and B. Nuseibeh, "Automatic detection of nocuous coordination ambiguities in natural language requirements," Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE 2010), pp.53–62, 2010.
[13] S. Liu, J. Sun, Y. Liu, Y. Zhang, B. Wadhwa, J.S. Dong, and X. Wang, "Automatic early defects detection in use case documents," Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE 2014), pp.785–790, 2014.
[14] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari, "Applications of linguistic techniques for use case analysis," Requirements Engineering, vol.8, no.3, pp.161–170, 2003.
[15] A. Sinha, S.M.S. Jr., and A. Paradkar, "Text2Test: Automated inspection of natural language use cases," Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST 2010), pp.155–164, 2010.
[16] W. Yu, J. Li, and G. Butler, "Refactoring use case models on episodes," Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE 2004), pp.328–335, 2004.
[17] J. Xu, W. Yu, K. Rui, and G. Butler, "Use case refactoring: A tool and a case study," Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC 2004), pp.484–491, 2004.
[18] V. Alves, R. Gheyi, and T. Massoni, "Refactoring product lines," Proceedings of the 5th international Conference on Generative Programming and Component Engineering (GPCE 2006), pp.201–210, 2006.
[19] K. Asano, S. Hayashi, and M. Saeki, "Detecting bad smells of refinement in goal-oriented requirements analysis," Workshop Proc. 36th International Conference on Conceptual Modeling (ER 2017), Lecture Notes in Computer Science, vol.10651, pp.122–132, 2017.
[20] E.R. Harold, Refactoring HTML: Improving the Design of Existing Web Applications, Addison-Wesley, 2012.
[21] L. Aversano, G. Canfora, G. De Ruvo, and M. Tortorella, "An approach for restructuring text content," Proceedings of the 35th International Conference on Software Engineering (ICSE 2013), pp.1225–1228, 2013.



Yotaro Seki received B.E. and M.E. degrees in computer science from Tokyo Institute of Technology in 2019 and 2021, respectively. His research interests include requirements engineering and natural language processing for requirements specification documents.

Shinpei Hayashi is an associate professor in School of Computing at Tokyo Institute of Technology. He received a B.E. degree in information engineering from Hokkaido University in 2004. He also respectively received M.E. and D.E. degrees in computer science from Tokyo Institute of Technology in 2006 and 2008. His research interests include software evolution and software development environment.

Motoshi Saeki received a D.E. degree in computer science from Tokyo Institute of Technology, in 1983. He was a professor in School of Computing at Tokyo Institute of Technology. He is currently a professor in Department of Software Engineering at Nanzan University. His research interests include requirements engineering, software design methods, software process modeling, and computer supported cooperative work (CSCW).