Prototype of a Supporting Tool to Generate Testing Communication Diagram

Tetsuro Katayama*, Seiya Urata*, Yohei Ogata*, Yoshihiro Kita†,
Hisaaki Yamaba*, Kentaro Aburada‡ and Naonobu Okazaki*
**University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192 Japan*
† Kanagawa Institute of Technology, 1030 Shimo-ogino, Kanagawa, 243-0292 Japan
‡ Oita National College of Technology, 1666 Maki, Oita, 870-0152 Japan
E-mail: kat@cs.miyazaki-u.ac.jp, urata@earth.cs.miyazaki-u.ac.jp, ogata@earth.cs.miyazaki-u.ac.jp,
y.kita@ccy.kanagawa-it.ac.jp, yamaba@cs.miyazaki-u.ac.jp, aburada@oita-ct.ac.jp, oka@cs.miyazaki-u.ac.jp

Abstract

This research has implemented a prototype of a supporting tool to generate testing communication diagram. The testing communication diagram helps a developer to understand where the software system is tested by a large quantity of test cases written in text, and it is generated by adding the information of test cases to communication diagram in UML (Unified Modeling Language). The implemented prototype can detect more efficiently deficiency and/or contradiction in communication diagram and/or test cases.

Keywords: Software development, Software testing, Test cases, Visualization, UML(Unified Modeling Language)

1. Introduction

In recent years, test cases used in software testing have become a larger scale as a software system becomes a larger. It is difficult to understand where the software system is tested by a large quantity of test cases written in text. Moreover, test cases or models to describe a software system with UML (Unified Modeling Language)[1] may have deficiency and/or contradiction because the work to design the test cases and to model the system is performed manually. It causes a situation that defects included in the system are not detected, leads to system failure after its operation, and gives users a great trouble.

The testing communication diagram has been proposed already.[2] It visualizes messages, which are written in a part of test cases, between objects in software system. It helps a developer to understand where the software system is tested by test cases written in text.

The testing communication diagram is generated by comparing test cases with communication diagram in UML and then adding the information of the test cases to the communication diagram. After the generation, a developer confirms whether or not deficiency and/or contradiction exist in communication diagram and/or test cases. Here, this process was gone manually.

In drawing testing communication diagram manually, one of problems is troublesome points to draw it and to confirm that the drawn diagram does not have any mistake. In confirming existence of deficiency and/or contradiction manually, one of problems is troublesome points to check it and to the checking process become more difficult as testing communication diagram becomes larger and more complex.

This research has implemented the prototype which can support to reduce the burden to generate testing communication diagram and to detect deficiency and/or contradiction in communication diagram and/or test cases. Because the prototype generates testing communication diagram automatically, it can reduce the burden to draw the diagram and it can rid the work to confirm that the drawn diagram does not have any mistake. And also, because highlight displays in a table of test cases and testing communication diagram are implemented as a function of the prototype, it can reduce the burden to detect deficiency and/or contradiction in communication diagram and/or test cases. Here, test cases are need to write in a template, a file format for test cases is CSV (Comma Separated Values), and a test suite means a collection of test cases.

2. Testing Communication Diagram

We explain steps to generate testing communication diagram.

(i) Select one of the test cases that are not yet in comparison with communication diagram.
(ii) Compare precondition with "participant name of sending side."
(iii) Compare input with "message name."
(iv) Compare precondition with "participant name of receiving side."
(v) When these agree, enclose the part represented an arrow of message in the diagram with a solid line. And, add a test case ID to inside the solid line.
(vi) If you have a test case that not yet compared, return to (i). Otherwise, testing communication diagram is completed.

The completed testing communication diagram supports to detect deficiency and/or contradiction in communication diagram and/or test cases by confirming whether arrows of messages is encircled, or whether each test case ID exists.

3. Prototype of a Supporting Tool to Generate Testing Communication Diagram

The implemented prototype has two major characteristics to reduce the burden to find deficiency and/or contradiction in communication diagram and/or test cases.

- Highlight display in a table of test cases.
  Background color of a cell in a table of test cases is highlighted red from white based on a comparison result between communication diagram and a CSV file in which test cases are written.
- Highlight display in testing communication diagram.
  Based on the comparison result, test case IDs are added each of message name in the communication diagram as "message name : test case ID", and their color is highlighted red from black.

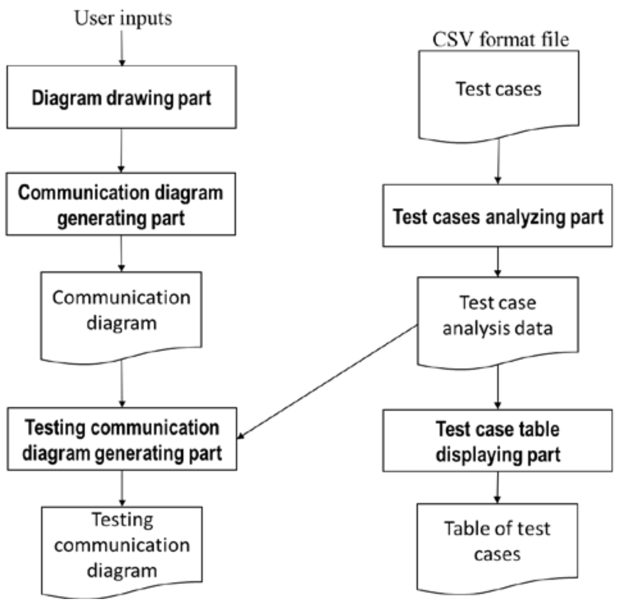

User inputs
Diagram drawing part
Communication diagram generating part
Communication diagram
Testing communication diagram generating part
Testing communication diagram
CSV format file
Test cases
Test cases analyzing part
Test case analysis data
Test case table displaying part
Table of test cases

Fig. 1. Structure of the implemented prototype

Fig. 1 shows structure of the prototype. The prototype consists of five parts: diagram drawing part, communication diagram generating part, test cases analyzing part, test case table displaying part, and testing communication diagram generating part.

The diagram drawing part has two panels: mode panel and paint panel, and supports a user to draw communication diagram. In the mode panel, the user can choose any objects the user want to draw. The paint panel send data to the communication diagram generating part. The data is both a type of the object generated by the user and the coordinates of the place clicked by the user.

The communication diagram generating part generates communication diagram by receiving the user's drawing requests through the diagram drawing part. It is regarded as communication diagram is completed when a request to generate objects from the diagram drawing part disappears.

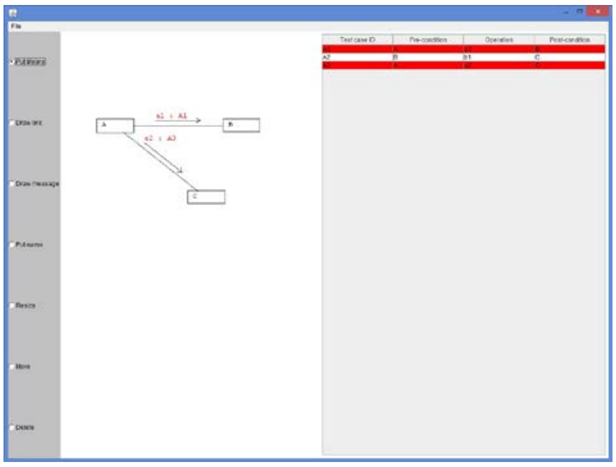
The test cases analyzing part generates test case analysis data. This part reads information by using *java.io.FileReader* class[3] and *java.io.BufferedReader* class[3] each line from the CSV file in which test cases are written along a form "test case ID, pre-condition, operation, post-condition". And then, it gets test case analysis data from the information by using *split* method of *java.lang.String* class[3]. The test case analysis data are generated by storing each string to arrays in order of test case ID, pre-condition, operation, and post-condition

The test case table displaying part generates a table of test cases. This part adds the array outputted from the test case analyzing part as one line of a table on the window. After finishing to add all elements of the array, this part compares test case IDs which is one of data stored to cells in the table with all test case IDs of each message object on the testing communication diagram. Here, a message object means one to express a message. The background color of cells is managed per a line of the table. The background color of the line where a comparison result is accorded is highlighted red from white. When all test case IDs are compared, this part displays a table of test cases.

The testing communication diagram generating part generates testing communication diagram. This part compares pre-conditions, operations, and post-conditions in the array stored test case analysis data with each attribute data of message objects in communication diagram. If all of them match after comparison, the font color of message name of its message object is highlighted red from black. In addition, its test case ID is added to the test case ID list which is an attribute of the message object. And then, this part generates testing communication diagram by redrawing each object on the window and highlighting message names and test case IDs. Here, the message name of the message object which has a test case ID is drawn as a form "message name : test case ID".

4. Overview of the Prototype

Fig. 2 shows an overview of the implemented prototype. The window consists has four parts: "File menu", "Mode panel", "Paint panel", and "Table of test cases". Each part is described as follows.



Test case ID	Pre-condition	Operation	Post-condition
A1	A	a1	B
A2	B	b1	C
A3	A	a2	C

Fig. 2. An overview of the implemented prototype

- File menu
  It is placed in the upper left corner of the window. If "open" is selected, a file dialog is displayed.
- Mode panel
  It is placed in the left of the window. In drawing communication diagram, each mode can be changed by user's click. At present, we have implemented seven modes: "Put lifeline", "Draw link", "Draw message", "Put name", "Resize", "Move", and "Delete".
- Paint panel
  It is placed in the middle of the window. It displays elements in communication diagram and testing communication diagram.
- Table of test cases
  It is placed in the right of the window. It is a table which displays test cases list.

We applied the prototype to some examples and confirmed that it worked properly. Elements which a comparison result is accorded in a table of test cases and testing communication diagram are highlighted red correctly. Moreover, elements which do not accorded do not highlighted.

In Fig. 2, test case ID A1 and A3 are highlighted and ID A2 is not highlighted.

5. Discussion

In this chapter, we discuss the usefulness of the implemented prototype.

We confirm the usefulness of our tool by experiments using examinees. The examinees draw testing communication diagram in hand by giving a communication diagram and test suits. For this work, it took an average of 232 seconds. In contrast, the time when the tool uses the same diagram and test suits as an input was an average of 5.2 seconds. Hence, the prototype can reduce 92% of the burden to generate testing communication diagram. Moreover, in chapter 4, we showed that our prototype generated testing communication diagram properly. Works to confirm that the generated testing communication diagram is not included human errors are not necessary because using our prototype prevents the human errors in generating testing communication diagram.

In addition, by the function of highlight display in a table of test cases and testing communication diagram, users can find deficiency and/or contradiction in them at glance. Hence, our tool can reduce the burden to detect deficiency and/or contradiction in communication diagram and/or test cases.

Some studies have been reported visualization of test results (e.g. Refs. 4) and some tools have functions of visualization of test results (e.g. Refs. 5). Also, research to automatically generate test cases from communication diagrams has been reported.[6] In contrast, our research attempts to visualize test cases which are usually described in text.

6. Conclusion

This research has implemented the prototype which can support to reduce the burden to generate testing communication diagram and to detect deficiency and/or contradiction in communication diagram and/or test cases. Because the prototype generates testing communication diagram automatically, it can reduce the burden to draw the diagram and it can rid the work to confirm that the drawn diagram does not have any mistake. And also, because highlight displays in a table of test cases and testing communication diagram are implemented as a function of the prototype, it can reduce the burden to detect deficiency and/or contradiction in communication diagram and/or test cases. It contributes to improve the software reliability.

Future issues as follows.

- Improvement in inputting communication diagram.
  To generate testing communication diagram with our prototype, users need to draw communication diagram on the prototype. To improvement of convenience of the users, the prototype will be accept diagram drawn on other tools.
- Improvement in displaying message names.
  In implementation of our tool, only one arrow in message is drawn. Because of this, plural message names are displayed parallel to the up of the arrow. We consider improvement to be easy to see the message name.
- Adaptation extension of comparable elements of communication diagram.
  At present, our prototype does not support communication diagram which has messages including loop condition. To support them, a part of the method to generate testing communication diagram will be reconstructed.
- Extension of a table of test cases.
  Test cases in this paper does not have expected outputs. We need to add a field to a table to describe the expected outputs.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 24500044.

References

1. UML Resource Page, http://uml.org/
2. S. Urata and T. Katayama, Proposal of testing diagrams for visualizing test cases, *Proc. 6th Int'l Conf. on Software Testing, Verification and Validation (ICST2013)* (2013).
3. Java™ Platform, Standard Edition 8 API Specification, http://docs.oracle.com/javase/8/docs/api/
4. J. A. Jones, Fault localization using visualization of test information, *Proc. 26th Int'l Conf. on Software Eng. (ICSE 2004)* (2004) 54-56.
5. Hertland.Data Inc., Dynamic Test tool DT10, http://hldc.co.jp/english/products/dt10/
6. P. Samuel, R. Mall, and P. Kanth, Automatic test case generation from UML communication diagrams, *Information and Software Technology, ScienceDirect,* **49**(2) (2007) 158-171.