Automatic Generation of Test Cases from Use-Case Diagram

Noraida Ismail [1*], Rosziati Ibrahim[2], Noraini Ibrahim[1]

[1]*Faculty of Information Technology and Multimedia,*
[2]*Research Management and Innovation Center (RMIC)*
University of Technology Tun Hussein Onn Malaysia (UTHM).
Parit Raja, Batu Pahat, Johor, Malaysia.

Intelligent searching techniques have been developed in order to provide a solution to the issue of finding information relevant to the user needs, and the problem of information overload - when far too much information is returned from a search. We employ this technique to introduce an automatic tool which used to generate the test cases automatically according to the system's requirement. The tool uses two steps for generating test cases. First, the system's requirements are transformed into a Unified Modeling Language (UML) Use-case Diagram. Second, the test cases will be automatic generated according to the use cases respectively. In the workspace, the ToolBox is used in order to ease the drawing of the use-case diagram. As well as allowing a user to layout the requirements of the system via a use-case diagram in the provided workspace, a user also may type-in the properties for each of the use cases used. Once the use-case diagram has been finalized, it can be save for further used and modification. The engine of the tool will take the use cases from the use-case diagram and search the query string (keyword) used in the tool's library. The searching engine uses both search keyword and additional information of the use-case diagram. This combination will result in improving data retrieval performance. Once the use case used matches the keyword inside the tool's library, the engine will automatically generate its respective test cases according to its use case.

Keywords: Intelligent Searching Engine, Artificial Intelligence, Information Retrieval, Automatic Generator, Use-Case Diagram

1. Introduction

Intelligent search technique has been proposed in order to soft the issue of information overload – when too much information is returned from a search (10). On the other hand, it is also help to provide a solution for reducing gap between what people really want to find, and the actual query strings they specify. This issue occurs because of same terms may have different meanings in different places. In order to overcome both problems, the information retrieval engine should intelligent enough to understand the user needs. The combination of query string specified and additional information will help the information retrieval engine to make a judgment for returning the likely relevance search result (7).

Therefore, we employ this technique into our tool, automatic generation of test cases from use case. This tool is used to generate the test cases automatically according to the system's use cases. These test cases are important to be used in analyzing and validating the requirements of the system. As an indispensable aspect in software development practices, software testing is important to reveal errors in the software and to ensure that software fulfills its requirements. In the traditional practices of software development lifecycle (SDLC), testing software is done at the later stage (analysis – design – prototyping – testing). From the empirical studies, delaying the software testing at later stage again will maximize the number of errors and make the process of fixing errors become complex, and this phenomenon will increase the budget of software development.

* Noraida Binti Ismail; anoraida@yahoo.com

This paper discusses on idea where this crucial part of SDLC is allowed to be done at early stages and proposes an automatic testing tool to validate either the system fulfills its requirements or not. This paper is organized as follows: several introductory related works are described in Section 2. Section 3 discusses the system's requirements. An idea on how to convert the use cases into test cases also will be given in Section 3. Section 4 discusses our tool in details, in particular on how to retrieve data from the database using the engine of the tool. In Section 5, we summarize our work and suggest future work.

2. Related Work

There are several discussions on how using use cases may help testing process to be done early in the development lifecycle. Jacobson et al. (4) explained tests can be derived from use cases in three types: First, tests of the expected flow of event; second, tests of unusual flow of events; and third, test of any requirements attached to a use case. Unfortunately, Jacobson et al. (4) did not discuss on how to choose test cases and how to know when you are done.

Binder (8), Heumann (6) and Wood et al. (3) derive test cases directly from requirements in natural language. Wood et al. (3) state that the most integral part of use case for generating test cases is the Event Flow (basic flow and alternate flow). The next step is creating the scenario based on the Event Flow before it can be used to generate the test cases.

However, automating the testing operation can reduce the cost and improve the reliability and effectiveness of software testing. Gutierez et al. (5) and Nebut et al. (2)

ISBN 978-979-16338-0-2

have showed how testing operation can be automated. Nebut et al. (2) study an approach for automating the generation of system test scenarios from use cases in the context of object oriented embedded software and taking into account traceability problems between high-level views and concrete test case execution.

In this paper, we are going to use use-case diagrams to automatically generate test cases. These test cases will become a checked list for software engineers in order to validate the system's requirement at the early software development stages.

3. The System's Requirements

In UML specification, requirements analysis and design are usually done using diagrams (1). One particular diagram (a use-case diagram) is used to specify requirements of the system. In a use-case diagram, two important factors are used to describe the requirements of a system. They are actors and use cases. Actors are external entities that interact with the system and use cases are the behaviour (or the functionalities) of a system (9) The use cases are used to define the requirements of the system. These use cases represent the functionalities of the system. Most often, each use case is then converted into a function representing the task of the system.

Therefore, we can convert from each of the use case into one test case or many test cases. The relationship of the conversion is either one to one or one to many. However, if we have many use cases, then we will have many test cases. Therefore, an automatic tool would be more wisely used in order to generate test cases from use cases of any system.

In most cases, use cases are developed based on the user perspective since the user is going to use the system. In order to make sure that the system does the requirements as it supposed to do, the test cases are designed according to the tester perspective. These test cases are basically designed to test the input and output of a system. Most often, the input, key-in by the user, will be accepted by the system. The system then processes the input and produces the required output according to its specification.

In this paper, we present an example of online bookstore system. The requirements of the system include the capability to make an order, cancel the order and check the status of his/her order. These three requirements are then transformed into a use-case diagram as shown in Figure 1.

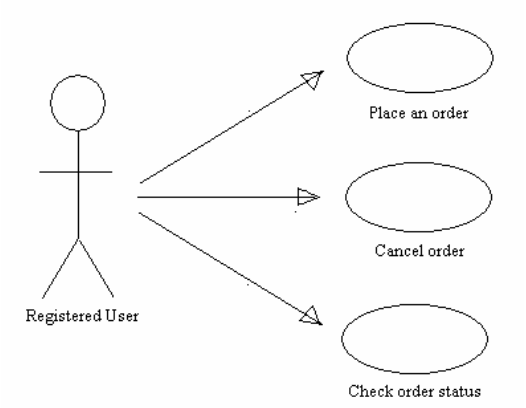

Figure 1: A Use-Case Diagram for an online bookstore system.

Figure 1 shows a simple use-case diagram for a online bookstore system where only a registered user (customer) is allowed to place an order for available items on this web. These registered users are also able to cancel the order that they make at the previous session. On the other hand, they may also check their order's status.

Most often, use cases represent the functional requirements of a system. If the requirements are gathered correctly, then a good use-case diagram can be formed. In UML, sequence diagrams are usually used to manually record the behaviour of a system by viewing the interaction between the system and its environment (5). These sequence diagrams describe in details activities for use cases. Therefore, the sequence diagrams can be used to help in generating the correct test cases. Based on Figure 1, a use-case diagram can be used to generate test cases of that particular system. But before test cases can be generated, the flow of events for each use cases must be defined first. As an example of one particular use case, Place an Order, the expected flow of events and its exceptions are shown in Table 1.

Table 1:Place an Order Event Flow

Place an Order Event Flow
1. User enters web site address in the browser. 2. User enters an email address and a password. Exception 1: email address and a password is not valid. • Log event • Use case ends 3. User enters search string – partial name of a book. Exception 2: No books matching search criteria were found • Log event • Use case ends 4. User selects a book. Exception 3: Decline a book • Log event • Use case ends

5. User adds the book to a shopping cart.

6. User selects "proceed to checkout" option.
   Exception 4: Continue shopping after storing a book in the shopping cart
   - Log event
   - Use case ends

7. User confirms shipping address.
   Exception 5: Enter a new address
   - Log event
   - Use case ends

8. User selects shipping option.

9. User confirms credit card that is stored in the system.
   Exception 6: Enter a new credit card
   - Log event
   - Use case ends

10. User places the order.
    Exception 7: Cancel order
    - Log event

    Use case ends

From Table 1, sequence diagrams are formed to record scenarios of the test cases. Based on Table 1 and test scenarios in sequence diagrams, we have derived the following test cases as shown in Table 2.

Table 2: Place an Order Test Case

Place an Order Test Case

Test Condition 1:
Basic flow of event – valid account/data is entered.
- An email address and a password.
- Matching search string were found
- Selects a book and add the book to a shopping cart
- Confirms shipping address and shipping option.
- Confirms credit card
- Places the order

Test Condition 2:
Email address and a password is invalid.
- Enter wrong combination of an email address and a password
- Enter unavailable of email address or password.
- Verify event is logged

Test Condition 3:
No books matching search criteria were found
- An email address and a password.
- Enter search string which is not spelled correctly.
- Verify event is logged.

Test Condition 4:
Decline a book
- An email address and a password.
- Matching search string was found.
- Enter a new search string.
- Verify event is logged

Test Condition 5:
Continue shopping after storing a book in the shopping cart
- An email address and a password.
- Matching search string were found
- Selects the book.
- Leave the book and select another book
- Verify event is logged

Test Condition 6:
Enter a new address
- An email address and a password.
- Matching search string were found
- Selects a book and add the book to a shopping cart
- Make a correction on the shipping address
- Change the shipping address.
- Verify event is logged

Test Condition 7:
Enter a new credit card
- An email address and a password.
- Matching search string were found
- Selects a book and add the book to a shopping cart
- Confirms shipping address and shipping option.
- Enter invalid credit card number.
- Change credit card number
- Verify event is logged

Test Condition 8:
Cancel order
- An email address and a password.
- Matching search string was found.
- Selects a book and add the book to a shopping cart.
- Confirms shipping address and shipping option.
- Confirms credit card.
- Click on cancel order button.
- Sign out of the account without place an order.
- Verify event is logged.

4. The tool

The tool, which we call GenTCase (Generator for Test Cases), can be used to layout the use-case diagram of any system. The tool is also able to automatically generate the test cases of the system according to the use-case diagram that has been formed previously. The tool is developed using object-oriented approach with C++ programming language. The tool has 3 major components as shown in Figure 2.

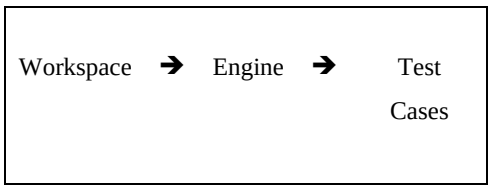

Figure 2: Components of GenTCase

From Figure 2, the tool allows a user to layout the use-case diagram of any system in the workspace provided. The workspace is used as a place for a user to provide the system's requirements by means of a use-case diagram. In

the workspace, a ToolBox is used to create, edit and display the use-case diagram. The ToolBox consists of standard symbols and arrows for a use-case diagram such as symbols for an actor and a use case, and arrows for connecting an actor with use cases as well as arrow for *generalizations*. In the Workspace, a user can also type-in the text for each of the use cases used in the *Text Box* provided by the tool. The Workspace will allow a user of the tool to layout the use-case diagram according to any system.

Once the use-case diagram has been finalized, the user can generate the test cases by using the generator of the tool. The Engine will take all the use cases and search the keywords used in the provided database. The database consists of most standard keywords of a use case. Once the use case used matches the keyword inside the database, the engine will generate its respective test cases according to its use case. Intelligent search technique is used to search all the metadata fields in the entire database.

The intelligent searching technique includes three major processes. First, the keywords are pre-processed by some automatic text operation methods. The result is a collection of metadata, which is considered the logical view of the use case diagram. Next, the metadata describing the logical views are used to construct a metadata-oriented index. An index such as this "allows fast searching over large volumes of metadata field".

During the retrieval, the information retrieval engine first performs similar text operations on the user query as those performed on the original use cases. The output of the text operation is a list of metadata, each of which is used to locate, through the index, a list of all the documents in which it occurs. When multiple metadata are present in the query, the search returns the union of the additional information retrieved by all the words. In short, searching is a process of matching keywords in the use cases with those in the query. Lastly, every retrieved metadata is evaluated by its relevance to the query and the additional information of use cases. The way the engine works is by choosing the shortest time-to-locate the object being searched. This will ensure the result returns in few seconds.

The tool will produce the test cases based on the use-case diagram provided in the workspace. These test cases are generated automatically from the tool as the output of the tool. The output is displayed on the screen as well as stored in a file with extension .txt, namely *output.txt*. A user can open this output file by using a NotePad or Microsoft Word. The output can be used as a checklist for a programmer to test the system that he or she will develop according to the provided test cases. These test cases can also be used to validate the results of the test cases so the requirements of the system are meet.

User who uses the tool can layout the use cases using the Workspace. The Tool Box is used in order to ease the drawing of the use-case diagram. Then, the button for generator of test cases (GTC) in the Workspace can be used to generate the test cases.

5. Conclusion and Future Work

GenTCase is a tool that is able to generate the test cases automatically according to the system's requirements. The test cases can be used as a checklist for a programmer to validate that the system meets its requirements. The purpose of GenTCase is to reduce the cost of testing the system. However, GenTCase has its limitations where the use cases used are only for functional requirements of a system. The tool is unable to capture the non-functional requirements of a system. Therefore, the non-functional requirements need to be captured and tested outside of the tool.

6. Acknowledgements

This research is under UTHM Fundamental Research Grant Vot 0233.

7. References

(1) A. Bahrami Object oriented systems development : using the unified modeling language, Mc-Graw Hill, Singapore. (1999)

(2) C. Nebut, F. Fleurey and Y.L. Traon, Automatic Test Geneartion: A Use Case Driven Approach, IEEE TRANSACTION ON SOFTWARE ENGINEERING Vol.32, No.3 (2003)

(3) D. Wood and J. Reis (1999). Use Case Derived Test Cases, Software Quality Engineering for Software Testing Analysis and Review (STAREAST99) Online. http://www.stickyminds.com/

(4) I. Jacobson, G. Booch, J. Rumbaugh. The Unified Software Development, England (1992)

(5) J. Gutierez, Escalona M.J. and Torres M.M. An Approach to Generate Test Cases from Use Cases, Proceedings of the 6th International Conference on Web Engineering. pp. 113-114 (2006).

(6) J. Heumann, Generating Test Cases from Use Cases, Rational Software, IBM. (2001).

(7) J. Jansen Using an Intelligent Agent To Enhance Search Engine Performance http://www.firstmonday.org (1996)

(8) R.V. Binder Testing Object-Oriented System. Addison-Wesley. USA (2000)

(9) Rational. (2003). Mastering Requirements Management with Use Cases, Rational Software, IBM.

(10) T. Stanley, Intelligent Searching Agent on the Web, http://ariadne.ac.uk/issue7/search-engine