Data mining: an automatic tool for retrieval of data from a system's requirements

R. Ibrahim[1], N. Ibrahim[2] & N. Ismail[2]
[1]Research Management and Innovation Center (RMIC), Universiti Tun Hussein Onn Malaysia (UTHM), Batu Pahat, Johor, Malaysia
[2]Faculty of Information Technology and Multimedia, Universiti Tun Hussein Onn Malaysia (UTHM), Batu Pahat, Johor, Malaysia

Abstract

Automatic information retrieval is usually used to ease the manual task of certain applications. In data mining, automatic information retrieval is used to retrieve specific data from a specific domain. We use this concept to introduce an automatic tool for data retrieval from requirements of a system, where the tool is used to generate the test cases automatically according to the system's requirements. The tool uses two steps for generating test cases. First, the system's requirements are transformed into a use-case diagram. Second, the use cases are then used to automatically generate the test cases. The tool allows a user to layout the requirements of a system via a use-case diagram in the workspace provided. 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 arrows for *extends* and *uses*. The workspace also allows a user to type-in the text for each of the use cases used. Once the use-case diagram has been finalized, it can be saved and edited at any other time. The engine of the tool will take all the use cases from the use-case diagram and search the keywords used in the provided database. Once the use case used matches the keyword inside the database, the engine will automatically generate its respective test cases according to its use case.
Keywords: information retrieval, automatic generator, use-case diagram.

1 Introduction

Data mining is the process of automatically searching large volumes of data for common knowledge discovery using tool [8]. The term data mining is often used to apply to two separate processes of knowledge discovery and prediction [7]. Knowledge discovery provides explicit information that has a readable form and can be understood by a user while prediction provides future events that may be transparent and readable in certain domain.

In data mining, automatic information retrieval is used to retrieve specific data from a specific domain. We use this concept to introduce an automatic tool for data retrieval from requirements of a system, where the tool is used to generate the test cases automatically according to the system's requirements. The generated test cases can be used as a checklist for a programmer or a developer of a system to validate that the system meets its requirements. The purpose of producing the tool is to reduce the cost of testing the system. System testing is important when developing a system. In many organizations, 30 to 50 percent of system development costs goes to testing the system [3]. Testing is defined as an attempt to reveal errors in a system. The conventional way of system development life cycle (SDLC - analysis, design, prototyping, testing) proposed that testing a system is done at a later stage. However, delaying the testing at a later stage will make the system testing cost more and expensive to correct the errors. In object-oriented system development (OOSD), system testing is proposed to be done as soon as the system's requirements are available. This will avoid postponing the system testing at the end of the system creation. Thus, it will minimize the errors and cost of fixing the errors.

This paper discusses on information retrieval for system testing where testing shows the present of faults and proposes an automatic testing tool to test whether a system meets its requirements or not. The rest of the paper is organized as follows. Section 2 presents the related work and Section 3 discusses the system's requirements. We also present our idea on how to convert the use cases into test cases 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. Finally, we conclude our paper in Section 5 and give some suggestions for future work of the tool.

2 Related work

In system testing, test cases are manually generated from use cases. Heumann [3] discusses on how to manually generate test cases from use cases using the basic flow of events and alternate flows of events. The scenarios are first created from combination of the basic flow and alternate flows. These scenarios are then used as the basis for creating test cases. Wood and Reis [9] presents an example on how to derive test cases and test scenarios from a use case. However, deriving test cases manually consumes more time and requires more tedious job. Therefore, many researchers study ways to automate the process of generating test cases, for example, Wee et al. [6], Hui and Hee [4] and Gutierez et al. [2]. Wee et al. [6] study the automation of test cases using the behaviour of object-oriented classes, where the test cases are automatically generated from the closed specifications of classes. They propose a scheme that combines the setup process, test execution and test validation into a single test program for testing the behaviour of object-oriented classes. The test program is generated automatically using the test cases and closed specifications of the classes.

Hui and Hee [4] study the validation of input and propose an approach for automated verification and test case generation of input validation from source code. Their work is concentrated on program source codes to get the input validation for generating the test cases. Gutierez et al. [2], on the other hand, propose an approach for automatic generation of test cases from use cases for web applications using activity diagrams representing the behavioural model of the system's requirements.

For our approach, instead of testing the software at the later stage, this research will use use-case diagrams to automatically generate the test cases. The test cases will then been analyzed in order to validate the requirements of the system. This will give more strength to our approach of using use-case diagrams for system specification, where ambiguity of the system's requirements will be reduced.

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 [5]. 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 an application for monitoring system of a postgraduate student submitting his/her progress report to Centre of Graduate Studies. The requirements of the system include the capability to submit progress report using the provided form, view the submitted progress report and evaluate the submitted progress report. These three requirements are then transformed into a use-case diagram as shown in Figure 1.

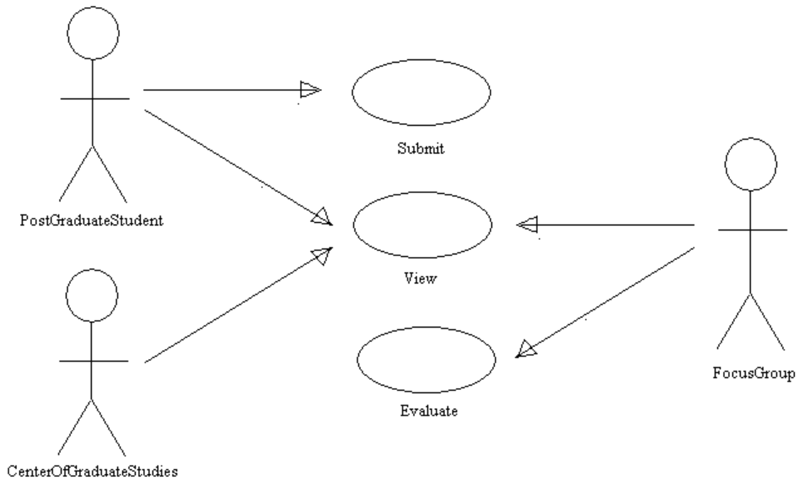


Figure 1: A use-case diagram for a monitoring system of a postgraduate student.

Figure 1 shows a simple use-case diagram for a monitoring system of postgraduate student where a postgraduate student (an actor) can submit his/her progress report to Centre of Graduate Studies. From Figure 1, a student is able to do two tasks: submit a progress report and view a progress report. A focus group is able to view and evaluate the progress report while the centre is able to view the progress report.

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. For example, Table 1 shows the test cases generated from the use case *Submit*.

Table 1: Test cases of use case *Submit.*

Use Case	Test Cases
Submit	Complete Progress Report Form is submitted
	Incomplete Progress Report Form is submitted
	No progress report is submitted

From Table 1, the generated test cases are the simplest test cases. From these test cases, sequence diagrams are formed to record scenarios of the test cases. These sequence diagrams will validate that the generated test cases confirm to its expected results. For example, when test case of incomplete progress report form is submitted, then the sequence diagram for that particular test case will check the consistency of its results. Figure 2 shows the sequence diagram for use case *Submit* for a scenario when an incomplete progress report is submitted.

4 GenTCase

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 3.

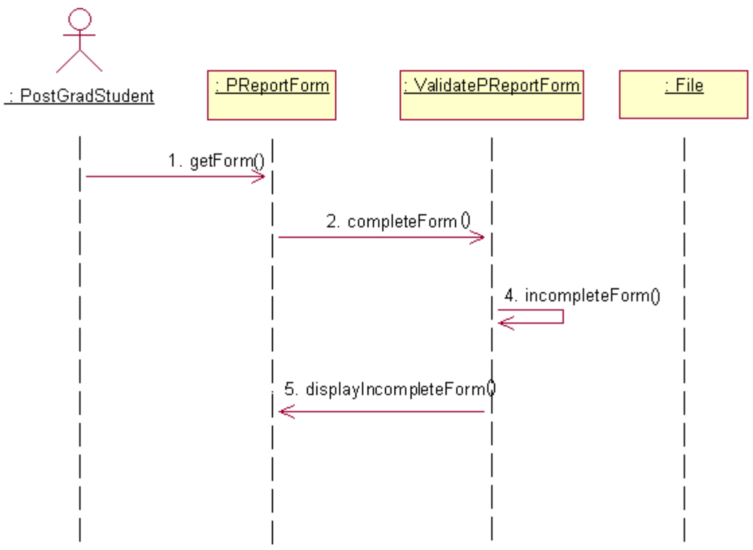


Figure 2: Sequence diagram for *Submit* when an incomplete form is submitted.

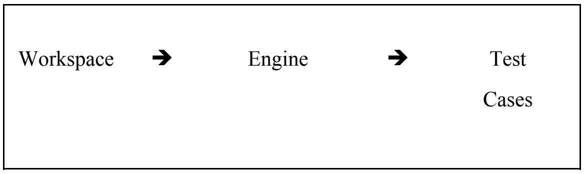


Figure 3: Components of GenTCase.

From Figure 3, 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. But instead of returning the long list of search results in which the search term happens to occur, the search engine returns a list of the metadata fields that satisfy the query. 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.

Figure 4 shows the tool for generating the test cases. User who uses the tool can layout the use cases using the workspace. The ToolBox is used in order to ease the drawing of the use-case diagram. The description of each of the button in the ToolBox is explained in Figure 5. Then, the button (as shown by number 11 for generator of test cases (GTC) in the workspace) can be used to generate the test cases.

Once the button of the *"Generator for the Test Cases"* is clicked, the test cases according to the use cases are generated accordingly. For example, Figure 6 shows the generated test cases according to use cases *Submit, View and Evaluate*.

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.

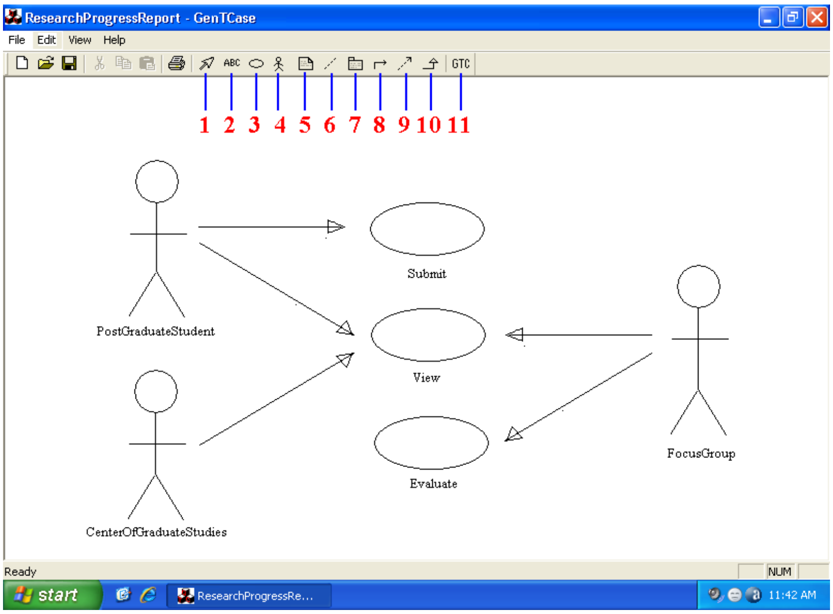
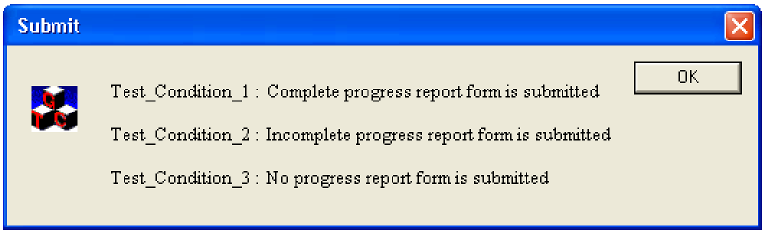


Figure 4: Interface of GenTCase for test cases.

No.	Item
1	Selection Tools
2	Text Box
3	Use Case
4	Actor
5	Note
6	Anchor Note
7	Package
8	Undirectional association
9	Dependency or instantiates
10	Generalizations
11	Generator for the Test Cases (GTC)

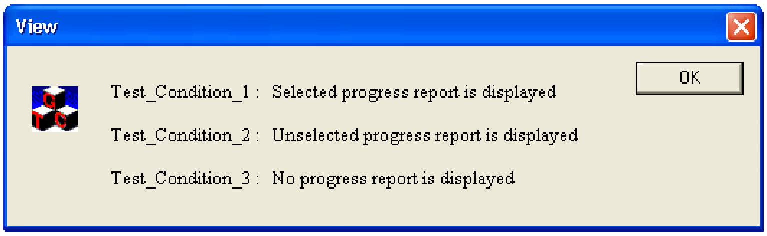
Figure 5: Descriptions of buttons in ToolBox.



Test_Condition_1 : Complete progress report form is submitted

Test_Condition_2 : Incomplete progress report form is submitted

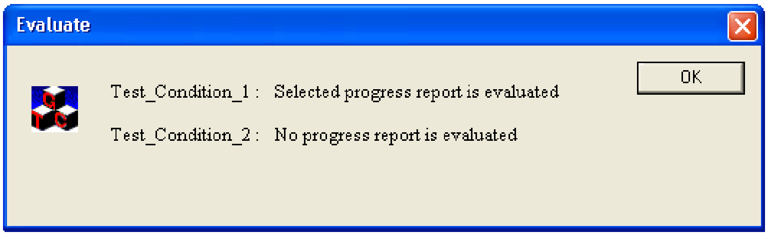
Test_Condition_3 : No progress report form is submitted



Test_Condition_1 : Selected progress report is displayed

Test_Condition_2 : Unselected progress report is displayed

Test_Condition_3 : No progress report is displayed



Test_Condition_1 : Selected progress report is evaluated

Test_Condition_2 : No progress report is evaluated

Figure 6: Test cases of use cases *Submit, View and Evaluate.*

Currently, GenTCase is able to generate test cases according to a single keyword for one use case. For example, a keyword *Submit* for use case will automatically generate three test cases. The process is repeated for every use case found in the use-case diagram. Therefore, the test cases will be generated according to use cases in the use-case diagram. We intend to extend GenTCase so it is able to accept more than one keyword for a single use case. For future enhancement of GenTCase, we also plan to include the flow of events (for the basic flow and alternate flows) of a use case so the test cases can be generated more efficiently and accurately according to the use cases.

Acknowledgement

This research is under UTHM Fundamental Research Grant Vot 0233.

References

[1] Bahrami A. (1999). *Object-Oriented Systems Development*, Mc-Graw Hill, Singapore.

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

[3] Heumann J. (2001). *Generating Test Cases from Use Cases*, Rational Software, IBM.

[4] Hui L. and Hee B.K.T (2006). *Automated Verification and Test Case Generation for Input Validation*, Proceedings of the 2006 International Workshop on Automation on Software Test (AST'06). pp. 29-35.

[5] Rational. (2003). *Mastering Requirements Management with Use Cases*, Rational Software, IBM.

[6] Wee K.L., Siau C.K. and Yi S. (2004). *Automated Generation of Test Programs from Closed Specifications of Classes and Test Cases*, Proceedings of the 26th International Conference on Software Engineering (ICSE'04). pp. 52-57.

[7] Wikipedia http://en.wikipedia.org/wiki/Main_Page

[8] Witten I. H. and Frank E. (2005). *Data Mining: Practical Machine Learning Toolsand Techniques*, 2nd Edition, Morgan Kaufmann.

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