AN OVERVIEW AND EVALUATION OF BLACK-BOX TESTING METHODS FOR SYSTEM TESTING

Anastasija Špakova and Marina Uhanova
Riga Technical University (RTU)
Riga, Latvia

ABSTRACT

The paper provides an overview of existing coverage-based functional black-box testing methods in order to choose testing method for checking student laboratory works in the study subject “Algorithmization and Programming of Solutions” at RTU Faculty of Computer Science and Information Technology. The aim is to increase the quality of the study subject and to automate student work checking process. It is needed to perform system type of testing in order to check student works. There are two similar testing methods that can help in checking works – scenario-based and domain testing. Nevertheless, the methods are similar and in theory should produce similar results, authors of the paper want to confirm if it is true or not. Practical application and evaluation of these methods will be presented in the paper.

KEYWORDS

Black-Box Testing Methods, Domain Testing Method, Scenario-Based Testing Method

1. INTRODUCTION

Testing can be described with five dimensions: testers, coverage, potential problems, activities and evaluation (Kaner et al. 2002). Each dimension gives tester direction for testing method selection. Testing methods can be involved in all five dimensions (Kaner et al. 2002). The focus of this paper is the two functional coverage-based testing methods which belong to coverage dimension: scenario-based and domain testing methods.

The system should be tested from customer’s perspective, and that appropriate test data to be selected for testing. Scenario-based testing checks the system from user perspective and domain testing helps to select appropriate test data. Both these methods are similar, thus in theory they should show similar results as well. Two metrics are used in order to clarify if this is true or not: number of defects and failure size – number of test cases in percentage in a set of test cases that have found a particular defect (Grindal et al. 2006).

The paper provides an insight about coverage-based testing methods. It will help to select appropriate testing method for student work checking in the study subject “Algorithmization and Programming of Solutions” in order to automate the checking process and to increase the quality of it.

The paper is organized as follows. Section 2 gives description of testing methods. Section 3 describes the testing program. Section 4 describes domain testing. Section 5 describes scenario-based testing. Section 6 gives examples of created test cases/test scenarios. Section 7 reviews the practical application of methods and summarizes the results. Section 8 includes conclusions and further research.

2. CLASSIFICATION OF COVERAGE-BASED TESTING METHODS

Several coverage-based testing methods are proposed in (Kaner et al. 2002):

- Function testing checks every feature or function which user can do in the system.
- Feature or function integration testing and scenario-based testing test several features or functions together in some combination.

- Use case testing checks the system on a transaction by transaction basis from start to finish (Tools QA).
 - Menu tour tests all of the menus and dialogs in a GUI product (Kaner et al. 2011).
 - Domain testing focuses on input/output values testing rather than the feature testing.
 - Equivalence class analysis tests input or output data which are divided in equivalence classes where only one test value of each class must be tested (Dustin 2002; Black et al. 2012).
 - Boundary testing tests input/output values that are located on equivalence class boundaries (Dustin 2002; Black et al. 2012).
 - Combination testing tests two or more input or output values in combination with each other.
 - Input field test catalogs test each type of input field.
 - Map and test all the ways to edit a field.
 - Logic testing tests logic or business rules. Input values have logical relationships in the system.
 - State-based testing or state transition testing checks if a system moves from state to state correctly.
 - Requirements-based testing checks if the system satisfies every requirement in a requirements document.
 - Specification-based testing tests the product specification.
- Coverage-based testing methods can be divided into three categories by their purpose (Table 1).

Table 1. Classification of coverage-based testing methods

Based on input/output values testing	Based on transactions testing (the sequence of actions)	Based on specific unit testing (system element or specific functionality)
<ul style="list-style-type: none"> • Domain testing • Equivalence class analysis • Boundary testing • Combination testing 	<ul style="list-style-type: none"> • Feature or function integration testing • Scenario-based testing method • Use case testing • Logic testing (Cause-effect graphing and decision table testing techniques) • State-based testing or state transition testing 	<ul style="list-style-type: none"> • Function testing • Menu tour tests • Input field test catalogs • Map and test all the ways to edit a field • Requirements-based testing • Specification-based testing

3. THE TESTING PROGRAM

State diagram is used for systematic test case creation. State diagram shows the states that a system can take and shows the events that cause and/or result from a change from one state to another (Black et al. 2012). In the case of domain testing, test cases are created by focusing on test data and in the case of scenario-based testing, test scenarios are created by focusing on various actions which are performed in the system. The program used for testing is one of the student's laboratory works written in Java. It is about order tracking for sewing manufactory. Program data: customer name/surname; date when ordered; order price; how much has already been paid. Program requirements: view file content; record order data; remove order data; adjust order data; order data by dates and surnames; find data about all customers who paid incompletely; calculate the total cost of all orders. The scheme of state diagram of task specification is shown in Figure 1.

Figure 1. State diagram for the program

4. DOMAIN TESTING METHOD

Domain testing considers numeric input and output values of fields. It implies that input/output values are divided into input/output domains of values. A domain is a set of values that includes all possible values of a variable of a function or feature. Domains with values are partitioned into sub-domains (equivalence classes) and then values from each sub-domain are used (Kaner et al. 2002; Kaner et al. 2005; Kaner et al. 2010). Two testing methods are used for domain testing: boundary value analysis and equivalence class analysis.

The structure of the domain testing which is used for our program testing is: identify input values and output values; partition input and output values into equivalence classes; define boundary values. The scheme of input values is shown in Figure 2. User selects any menu option. If user selects “data about new order” or “data editing”, then he needs to enter/change respective input data. Format and value type (numeric, non-numeric) should be checked. Output values are shown schematically in Figure 3. User selects any menu option and the result (output) of this option is saved in the file. File format and content should be checked. Equivalence partitions and boundary values of input/output values are shown in Table 2.

Figure 2. Input values of the program

Figure 3. Output values of the program

Table 2. Equivalence partitions and boundaries of input and output values

Input/output values	Valid equivalence classes	Invalid equivalence classes	Valid boundaries	Invalid boundaries
Menu option selection (input)	Correctly selected	Incorrectly selected	Not applicable	Not applicable
Name (input)	Non-integer	Integer; empty	Not applicable	Not applicable
Surname (input)	Non-integer	Integer; empty	Not applicable	Not applicable
Date when ordered (input)	Correct format	Non-integer; incorrect format; date in the past; empty	1.06.2020; 15.12.2020	Not applicable
Order price (input)	0.00 – Max (infinity); integer; correct format; >how much has already been paid	<0.00; non-integer; incorrect format; empty; <how much has already been paid	0.00; 999.99; order price = how much has already been paid; order price = how much has already been paid + 0.01	– 0.01; order price = how much has already been paid – 0.01
How much has already been paid (input)	0.00 – Max (infinity); integer; correct format; <order price	<0.00; non-integer; incorrect format; empty; >order price	0.00; 999.99; how much has already been paid = order price; how much has already been paid = order price – 0.01	– 0.01; how much has already been paid = order price + 0.01
File output when removing data about order from the file	There are orders to remove	There are no orders to remove	Not applicable	Not applicable
File output when ordering by dates and customer surnames	Not applicable	Not applicable	01.01.2020; 15.12.2021; date + 1 day; date – 1 day; date 1 = date 2 (two identical); customer surname 1 = customer surname 2 (two identical)	Not applicable
File output when searching customers who paid incompletely	There are customers; there are no customers	Not applicable	Not applicable	Not applicable
File output when calculating total cost of all orders	0.00 (all orders); 1 order – Max	Not applicable	0, 1 and 2 orders	Not applicable
File format (created file is output)	Correct file format	Incorrect file format	Not applicable	Not applicable

5. SCENARIO-BASED TESTING METHOD

Scenario-based testing method deals with creation of test scenarios. Test scenario is based on a scenario that is a hypothetical story or situation about system use (Kaner et al. 2011). The structure of the scenario-based testing which is used for our program testing: identify all features which user can do in the system and identify all situations how user can use the feature. Feature and user actions are the following:

1. **View file content (option “View”).** User views content when file is empty and when it contains orders.
2. **Record data about new order (option “Record”).** User enters data about customer; records new order directly in the file; enters all fields incorrectly; adjusts one field incorrectly and other fields correctly.
3. **Remove data about order from the file (option “Remove”).** User removes order; enters removed order once again and removes it; tries to remove order when no orders are in the file; adjusts all fields incorrectly; adjusts one field incorrectly and all other fields correctly.
4. **Adjust order data (option “Adjust”).** User adjusts data about customer using option “Adjust”; adjusts data directly in the file; leaves one field empty and enters other fields correctly.
5. **Order data by dates and surnames (option “Order”).** User orders by dates and surnames.
6. **Find data about all customers who paid incompletely (option “Find”).** User searches customers who paid incompletely.
7. **Calculate total cost of orders (option “Calculate”).** User calculates cost when there are 0, 1, 10 orders.
8. **Exit the program (option “Exit”).** User selects menu option “Exit” after he opened a program; viewed the file content; recorded, removed and adjusted order; ordered data by dates and surnames; found data about all customers who paid incompletely for work; calculated the total cost of all orders.

6. CREATION OF TEST CASES AND TEST SCENARIOS

Domain and scenario-based testing methods can overlap. Boundary values/equivalence classes are used for test values selection in scenario-based testing method. For boundary values/equivalence classes selection in domain testing the tester must think about various user actions what user might want to do in the system. For example, one of user actions can be that he can enter new order correctly when order price is 999.99, and this is boundary value which is identified using domain testing. Examples are shown in Table 3 and Table 4.

Table 3. Test case “User enters new order incorrectly”

Precondition: The file should be empty.	
Steps	Expected results
1. Select menu option “Record”.	The screen for data entering is shown.
2. Enter the following data: name = integer value; surname = integer value; date when ordered = 00:00; order price = – 0.01; how much has already been paid = – 0.01. Press “Save”.	Error is shown that user entered data incorrectly.
3. Select menu option “View”.	No data are shown.
4. Open created file and check if previously entered data are saved in the file.	Previously entered data are not saved in the file.

Table 4. Test scenario “User enters new order incorrectly”

Steps	Expected results
1. Select menu option “Record”.	The screen is shown.
2. Select “Exit”.	User exited the program.
3. Select menu option “Record”.	The screen is shown.
4. Enter the following data: name = non-integer value; surname = non-integer value; date when ordered = 00:00; order price = 999.99; how much has already been paid = 0.01. Press “Save”.	The data are saved.
5. Select menu option “View”.	User can see the data.
6. Select “Exit”.	User exited the program.
7. Open created file and check if previously entered data are saved in the file.	Data are saved in the file.

7. TEST EXECUTION AND RESULTS

21 test cases and 28 test scenarios were created. Test cases/test scenarios were executed manually. Domain testing and scenario-based testing found the same defects. 5 defects were found altogether.

The more test cases/scenarios find one and the same defect, the higher is possibility to find this defect (Grindal et al. 2006). Table 5 and Table 6 shows failure sizes. Test results show that the highest number of test cases/test scenarios that is unsuccessfully executed due to one and the same defect is 10 test cases and test

scenarios for both testing methods. The 1st defect is found using domain testing method, but scenario-based testing did not find it. The probability to find the 5th defect is bigger for scenario-based testing than for domain testing. Conclusion can be made that both methods are good for finding defects and each method concentrates on specific aspect of testing. Both methods also can find one and the same defect.

Table 5. Failure size (Domain testing)

Defect No.	Number of test cases affected by the defect	Failure size
1.	1	4,76%
2.	10	47,62%
3.	1	4,76%
4.	2	9,52%
5.	2	9,52%

Table 6. Failure size (Scenario-based testing)

Defect No.	Number of test scenarios affected by the defect	Failure size
1.	0	0%
2.	10	35,71%
3.	1	3,57%
4.	2	7,14%
5.	4	14,29%

8. CONCLUSIONS AND FURTHER RESEARCH

Both evaluated methods – domain testing and scenario-based testing produce similar results and can find one and the same defects because the methods overlap. The statement that similar methods produce similar results is approved by the testing results. This means that there are no good or bad testing methods. Each method focuses on specific aspect of testing. Tester should decide what is relevant in the specific situation – to use only one method or to use combination of several methods. Both evaluated methods can be applied for student laboratory work checking and can effectively find defects. The authors of this paper suggest to use domain testing and scenario-based testing in combination in order to gain more thorough testing and possibly to find more defects.

The program tested was relatively small so further practical evaluation of these two methods should be performed on more complex and bigger systems. The amount of test cases/scenarios depends on size of the system and it will increase in case of bigger system testing. Thus, automation of test cases/scenarios should be applied in that case. The further research is to create automatic tests using these or other functional black-box testing methods. For evaluation of these methods it is required to select structural testing test coverage criteria (e.g. statement, decision coverage) in order to know how much each method can cover the code.

REFERENCES

- Black, R. et al, 2012. *Foundations of software testing: ISTQB certification*. 3rd Edition, Cengage Learning EMEA, Hampshire.
- Dustin, E., 2002. *Effective Software Testing: 50 Specific Ways to Improve Your Testing*. Addison-Wesley.
- Grindal, M., et al, 2006. *An Evaluation of Combination Strategies for Test Case Selection*. Available at: http://barbie.uta.edu/~mehra/54_An%20Evaluation%20of%20Combination%20Testing%20Strategies.pdf (Accessed: 31 March 2019).
- Kaner, C. and Bach, J., 2005. *Black Box Software Testing 2005 Academic Edition Domain Testing*. Available at: <http://testingeducation.org/BBST/domain/BBSTDomainTesting1.pdf> (Accessed: 31 March 2019).
- Kaner, C. and Fiedler, R.L., 2011. *Black Box Software Testing: Introduction to Test Design: A Survey of Test Techniques*. Available at: <http://www.testingeducation.org/BBST/testdesign/BBSTTestDesign2011pfinal.pdf> (Accessed: 31 March 2019).
- Kaner, C. and Padmanabhan, S., 2010. *An introduction to the theory and practice of domain testing*. Available at: <http://kaner.com/pdfs/kanerPadmanabhanVISTACONDomainTesting.pdf> (Accessed: 31 March 2019).
- Kaner, C., et al, 2002. *Lessons Learned in Software Testing: A Context-Driven Approach*. 1st Edition, John Wiley & Sons, Inc., New York.
- Tools QA. Available at: <http://toolsqa.com/software-testing/test-scenario/> (Accessed: 28 March 2019).