International Journal of Multidisciplinary Research and Development
Online ISSN: 2349-4182 Print ISSN: 2349-5979
www.allsubjectjournal.com
Volume 3; Issue 2; February 2016; Page No. 21-23

Effective testing techniques

Deepika Bhagat

Assistant Professor, Chandigarh Group of Colleges, Landran, Punjab, India.

Abstract
By design we mean to create a plan for how to implement an idea and technique is a method or way for performing a task. So, test design is creating a set of inputs for given software that will provide a set of expected outputs. The idea is to ensure that the system is working well enough and it can be released with as few problems as possible for the average user. This not only saves cost, time and labor of the company but also raises the good will of the company. There are many testing techniques obtainable. The choice of an effective and cost saving testing technique has always been a hunt for the developers. The paper attempts to provide a comprehensive view of Testing Techniques. The objective is to put all the relevant information into a unified context.

Keywords: Equivalence Partitioning, Boundary Value Analysis, Cause-Effect Graphing Techniques, Decision Table Testing, Error Guessing.

Equivalence Partitioning
This method divides the input domain of a program into classes of data from which test cases can be derived. It is based on an evaluation of equivalence classes for an input condition. An equivalence class represents a set of valid or invalid states for input conditions. Equivalence classes may be defined according to the following guidelines:

1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
2. If an input condition requires a specific value, then one valid and two invalid equivalence classes are defined.
3. If an input condition specifies a member of a set, then one valid and invalid equivalence class is defined.
4. If an input condition is Boolean, then one valid and one invalid equivalence class are defined.

Grocery Store Example: Consider a software module that is intended to accept the name of a grocery item and a list of the different sizes the item comes in, specified in ounces. The specifications state that the item name is to be alphabetic characters 2 to 15 characters in length. Each size may be a value in the range of 1 to 48, whole numbers only. The sizes are to be entered in ascending order (smaller sizes first). A maximum of five sizes may be entered for each item. The item name is to be entered first, followed by a comma, and then followed by a list of sizes. A comma will be used to separate each size. Spaces (blanks) are to be ignored anywhere in the input.

Derived Equivalence Classes

1. Item name is alphabetic (valid)
2. Item name is not alphabetic (invalid)
3. Item name is less than 2 characters in length (invalid)
4. Item name is 2 to 15 characters in length (valid)
5. Item name is greater than 15 characters in length (invalid)
6. Size value is less than 1 (invalid)
7. Size value is in the range 1 to 48 (valid)
8. Size value is greater than 48 (invalid)
9. Size value is a whole number (valid)
10. Size value is a decimal (invalid)
11. Size value is numeric (valid)
12. Size value includes non-numeric characters (invalid)
13. Size values entered in ascending order (valid)
14. Size values entered in non-ascending order (invalid)
15. No size value entered (invalid)
16. One to five size values entered (valid)
17. More than five sizes entered (invalid)
18. Item name is first (valid)
19. Item name is not first (invalid)
20. A single comma separates each entry in list (valid)
21. A comma does not separate two or more entries in the list (invalid)
22. The entry contains no blanks (???)
23. The entry contains blanks (????)

Table 1: Black Box Test Cases for the Grocery Item Example based on the Equivalence Classes Above.

#	Test Data	Expected Outcome	Classes Covered
1	Xy,1	T	1,4,7,9,11,13,16,18,20,22
2	AbcDefghijklmno ,1,2,3,4,48	T	1,4,7,9,11,13,16,18,20,23
3	A2x,1	F	2
4	A,1	F	3
5	abcdefghijklmonp	F	5
6	Xy,0	F	6
7	XY,49	F	8
8	Xy,2.5	F	10
9	xy,2,1,3,4,5	F	14
10	Xy	F	15
11	XY,1,2,3,4,5,6	F	17
12	1,Xy,2,3,4,5,6	F	19
13	XY2,3,4,5,6	F	21
14	AB,2#7	F	12

Boundary Value Analysis: Boundary value Analysis: - Boundary value analysis is the technique of making sure the behavior of system is predictable for the input and output boundary conditions. Reason why boundary conditions are

very important for testing is because defects could be introduced at the boundaries very easily. It's widely recognized that input values at the extreme ends of input domain cause more errors in system. Most application error occurs at the boundaries of input domain. 'Boundary value analysis' testing technique is used to identify error at boundaries rather than finding those exist in center of input domain. Boundary value analysis is a next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.

Boundary Value Analysis Example: If you are testing for an input box accepting number from 1 to 1000 then there is no use in writing thousand test cases for all 1000 valid input numbers plus other test cases for invalid data. Test cases for input box accepting numbers between 1 and 1000 using Boundary value analysis:

1) Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 1000 in our case.
2) Test data with values just below the extreme edges of input domains i.e. values () and 999.
3) Test data with values just above the extreme edges of input domain i.e. values 2 and 1001.

Note: There is no hard-and-fast rule to test only one value from each equivalence class you created for input domains. You can select multiple valid and invalid values from each equivalence class according to your needs and previous judgments. E.g**.** If you divided 1 to 1000 input values in valid data equivalence class, then you can select test cases values like: 1,11,100,950 etc. same is the cases for other test cases having invalid data classes.

Cause effect Graphing Technique: A cause-effect graph is a directed graph that maps a set of effects. The causes may be thought of as the input to the program, and the effects may be thought of as the output. Usually the graph shows the nodes representing the effects on the right side. There may be intermediate nodes in between that combine inputs using logical operators such as' AND' and 'OR'. The cause-Effect Graphing technique was invented by Bill Elmendorf of IBM in 1973. There are four steps:

- Causes (input conditions) and effects (actions) are listed for a module and an identifier is assigned to each.
- A cause-effect graph is developed.
- The graph is converted to a decision table.
- Decision table rules are converted to test cases.

Once the test cases have been derived by the test cases designer, the process include the review of these test cases by the following stakeholders:

The author of the requirements verifies that he/she agrees with the test cases designer's translation of requirement to test cases.

The domain experts review the test cases in order to determine the answer to the following question: "Are we building the right system"?

The developers review the test cases to clarify their understanding of the requirements. The developers learn what they will be tested on, and can therefore develop the software to succeed. The cause-effect graph that represents this requirement is provided in the figure below. The cause-effect graph shows the relationship between the causes and effects.

Fig 1

In Figure above A, B & C are called nodes, A and B are the causes, while node C is an effect. Each node can have a true or false Condition. The lines, called vectors, connect the cause nodes A and B to the effect node C. All requirements are translated into nodes and relationships on the cause-effect graph. There are only four possible relationships among nodes, and they are indicated by the following symbols:

- Where A always leads to C, a straight line ----------.
- Where A or B lead to C, a V at the intersection means "OR".
- Where A or B lead to C, an inverted V at the intersection means "and".
- A tilde ~ means "not" as in "if not A, then C".

The cause-effect graph is then converted into a decision table or "truth table" representing the logical relationships between the causes and effects. Each column of the decision table is a test case. Each test case corresponds to a unique possible combination of inputs that are either in a true state, a false state.

Decision Table Technique

Decision Table Testing involves testing the behavior of a system when any component involves logical conditions in it and system needs to be verified for different combinations of the conditions. Internal behavior of the system and its design involving logical conditions can be easily captured and documented with decision tables testing. Testers need to analyze the specification and then identify the conditions and actions of the system to capture them in decision tables. The input condition and actions are most often stated in such a way that they can either be true or false (Boolean). The decision table contains the triggering conditions, often combination of true and false for all input conditions, and the resulting actions for each combination of conditions. The advantage of decision table testing is that tester verifies all the combinations of conditions which might get skipped during normal testing. For example: - A product sends the notification to a user based on user's preferences ONLY provided notification are turned ON from Admin interface. This involves verifying the feature for different combinations as mentioned below.

Table 2

Setting at admin Level	Setting at User Level	Final Result
OFF	OFF	System should not send the notification to the user.
ON	OFF	System should not send the notification to the user.
OFF	ON	System should not send the notification to the user.
ON	ON	System should send the notification to the user.

Error Guessing: Error Guessing involves making an itemized list of the errors expected to occur in a particular area of the system and then designing a set of test cases to check for these expected errors. It is more of testing art than testing science but can be very effective, given a tester familiar with the history of the system. It involves asking "what if?"
Example: The statement might be made "the user must input a valid zip code". What if the user enters no zip code? What if the zip doesn't match the state? What if the zip code doesn't exist? Etc. it comes with experience with the technology and the project. It is the art of guessing where errors can be hidden. There are no specific tools and techniques for this, but you can write test cases depending on the situation: Either when reading the functional documents or when you are testing and find an error that you have no documented. It is a test case technique to check the application with invalid data, whether it is accepting the data or not.

Summary: Equivalence partitioning strives to define a test case that uncover classes of errors and thereby reduces the number of test cases needed. It is based on an evaluation of equivalence classes for an input condition. Boundary value analysis is the technique of making sure the behavior of system is predictable for the input and output boundary conditions. A cause-effect graph is a directed graph that maps a set of causes to a set of effects. Decision Table Testing involves testing the behavior of a system when any component involves logical conditions in it and system needs to be verified for different combinations of the conditions. Error Guessing involves making an itemized list of the errors expected to occur in a particular area of the system and then designing a set of test cases to check for these expected errors.

References

1. Introduction to SQA & Testing Vol. I by BTES
2. Lessons Learned in Software Testing, by Kaner C, Bach J, Pettichord B.
3. Testing Computer Software, by Kaner C, Falk J, Nguyen H.
4. Software engineering, by Roger Pressman
5. http://www.diffen.com/
6. http://istqbexamcertification.com/what-is-test-design-technique/
7. http://www.ncbi.nlm.nih.gov/pmc/
8. http://www.iso.org/iso/home/standards/management-standards/iso_9000.html