Volume 3, Issue 9, September 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Software Testing Techniques for Test Cases Generation

Gaurav Saini[1],
[1]Research Scholar,
Department of CSE,
Chandigarh Engineering College, India

Kestina Rai[2]
[2]Assistant Professor,
Department of CSE,
Chandigarh Engineering College, India

Abstract— Software testing is very important prospective in various product accuracy. Software testing is set of activities conducted with the intent of finding errors in software. It also verifies and validate whether the program is working correctly with no bugs no not. There are basically three levels of testing- Unit, Integration and System. Unit testing referred to as testing in small whereas Integration and System testing are referred to as testing in large. Various testing techniques available for designing of test cases. This paper basically deals with various techniques available to design software testing test cases.

Keywords— Software testing, Unit, Integration, System Testing, Equivalence Partitioning, Boundary Value Analysis, Error guessing, Acceptance Testing, Testing Sequence.

I. INTRODUCTION

For a software project, usually around 50% of money and resources go under software testing. The IEEE definition of testing is "the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify difference between expected result and actual result.

Expected result: The final outcome defined is based on requirements specifications of the test execution.
Actual result: The actual outcome received after applying the test data to the software/ feature as per steps defined in test case under controlled test environment.

Software testing is the process of executing a software system to determine whether it matches its specifications and executes in its intended environments. A software failure occurs when a piece of software does not perform as required and as expected. In software testing, the software is executed with input data or test cases and output of the software is observed. The testing process can be divided into three steps- test case generation, test case execution and test evaluation. A test case is the triplet [I, S, O], where I is the input given to the system and S is the state of the system at which input is given and O is the expected output of the system.

Levels of testing: As we know that each product development starts with development of functional requirements specifications and then the product goes into the development phase. Each product is broken into multiple smaller modules/phases so as to facilitate the development process and easily track the progress and above all reduce the risk of failures during the development process. Development of the product also happens in different levels starting from creation of components, components integration and then complete system development. Testing starts from testing a simple screen, sub-modules, module, combination of modules, and then finally testing the complete system.

A. *Unit testing:* Unit testing focus on the verification effort of the smallest unit of software design-the unit. The units are identified at the detailed design phase of the software development life cycle and the unit testing can be conducted in parallel for multiple units.

B. *Integration testing:* After unit testing, modules shall be assembled or integrated to form the complete software package as indicated by the high level design. Integration testing is systematic technique for verifying the software structure and sequence of execution while conducting tests to uncover errors associated with interfacing. Black-box test case design techniques are most prevalent during integration, although limited amount of white-box testing may be used to ensure coverage of major control paths. Integration testing is sub-divided as follows:
   - *Big-bang approach.*
   - *Top-down approach*
   - *Bottom-up approach*
   - *Sandwich (Mixed integration) approach.*

C. *System testing:* After all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that developed system conforms to its requirements laid out in the SRS document.

D. Acceptance testing: when the customized software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all the requirements. Acceptance tests are conducted at the development site or at the customer site depending upon the requirements and mutually agreed principles. Acceptance testing may be conducted either by the customer depending upon the type of project and the contractual agreement. Alpha nd Beta testing also forms of acceptance testing.

Alpha testing: Alpha testing is the system testing performed by the development team.
Beta testing: Beta testing is the system testing performed by the customer himself after the product delivery to determine whether to accept the delivered product or to reject it.

II. SOFTWARE TESTING TECHNIQUES

The purpose of software testing is to identify all the defects in a program. There are many techniques available for designing of test cases as given below.

A. Equivalence Partitioning: This method divides the input domain of a program into classes of data from which test cases can be derived. Equivalence partitioning strives to define a test case that uncovers 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. An equivalence class represents a set of valid or invalid states for input conditions. Equivalence class 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 one invalid equivalence classes are defined.
4. If an input condition is Boolean, then one valid and one invalid equivalence classes are defined.

Test case design for Equivalence partitioning:

- Good test case reduces by more than one the number of other test cases which must be developed.
- Goog test case covers a large set of other possible cases.
- Classes of valid inputs.
- Classes of invalid inputs

Equivalence Class partitioning 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 (smallest size 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 of 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 orders (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 GROCERY TIEM EXAMPLE BASED ON THE EQUIVALENCE CLASSES ABOVE:

Sr. No.	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	abcdefghijklmnop	F	5
6	X,y,0	F	6
7	XY,49	F	8
8	Xy,2,5	F	10
9	xy,2,1,3,4,5,6	F	14
10	Xy	F	15
11	XY,1,2,3,4,4,6	F	17
12	1,Xy,2,3,4,5	F	19
13	XY2,3,4,5,6	F	21
14	AB,2#7	F	12

B. *Boundary Value Analysis:* Boundary value analysis is the technique of making sure that behaviour of system is predictable for the input and output boundary conditions. Reason why boundary conditions are very important for testing because defects could be introduced at the boundary very easily. It's widely recognized that input values at the extreme ends of input domain cause more errors in system. More applications errors occurs at the boundaries of input domain. 'Boundary value analysis' testing technique is used to identify errors at boundaries rather than finding those exist in centre of input domain. Boundary value analysis is the 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 u are testing for an input box accepting numbers 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:

- Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 1000 in our case.
- Test data with values just below the extreme edges of input domain i.e. values 0 and 999.
- Test data with values just above the extreme edges of input domain i.e. values 2 and 1001.

E.g. if you divided 1 to 1000 input values in valid data equivalence class, then you can select test case values like: 1,11,100, 950 etc. Same is the case for other test cases having invalid data classes.

C. *Cause effect Graphing Technique:* A cause effect graph is directed graph that maps a set of causes to 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 causes on the left side and 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.

The cause-effect graphing technique begins with the set of requirements, and determines the minimum number of test cases to completely cover the requirements.

Test Case Review:

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

- The author of the requirements verifies that he/she agrees with the test case designer's translation of requirements to test cases.
- The domain experts review the test cases in order to determine the answer to the question: "Are we building the right system".

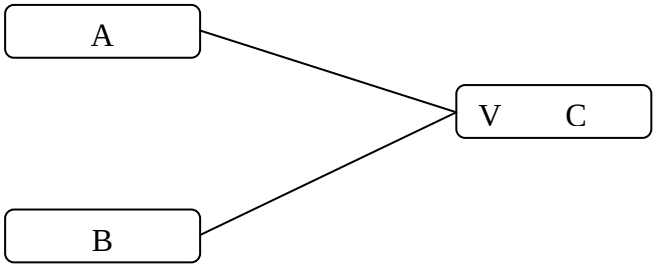
- The developers review the test cases to clarify their understanding of the requirements. The developers learn what they will be tested on, and can therefore develop the software to succeed.

By performing these reviews, everyone on the project team can obtain the same understanding of what will be built.

Cause-Effect Graphing Example:
A requirement says: "If A OR B, then C". The following rules hold for this requirement:

- If A is true and B is true, then C is true.
- If A is true and B is false, then C is true.
- If A is False and B is true, then C is true.
- If A is false and B is false, then C is False.

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.



In figure above, A, B and C are called nodes. 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 leads to C, a V at the intersection means "or".
- Where A and B leads 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 input that are either in a true state, a false state.

D. Decision table technique: Decision table technique involves testing the behaviour of a system when any component involves logical conditions in it and system needs to be verified for different combinations of the conditions. Internal behaviour of the system and its design involving logical conditions can be easily captured and documented with decision table testing. Testers need to analyse the specifications and then identify the conditions and actions of the system to capture them in decision tables.

The input conditions 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 combinations of true and false for all input conditions, and then 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 notifications are turned ON from Admin interface. This involves verifying the feature for different combinations as mentioned below:

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.

E. Error Guessing Technique: 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 code 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 upon the situation. Either when reading the functional documents or when you are testing and find an error that you have not documented. It is a test case technique to check the application with invalid data, whether it is accepting the data or not.

III. TESTING SEQUENCE

A tester is the nearest person to the real-time end user of the product who will be using the product. He acts like virtual end user who will use the product in real-time scenarios. Tester needs to understand not just his/her features, but also other features in the product which interact with his/her features, thus a tester knows whole of the product under testing. The testing sequence is given below:

- **Functionality:** This is the first thing to test in a feature under test because this is what the users are paying for. For example: - Test if you can save a file on a specific location and verify the file's existence on that location.
- **GUI:** Verify all the fields on the graphical user interface for data validation of those fields. In early stage of product testing, these errors are very much existent and you can easily find them.
- **Error Conditions:** Test the software to verify if it handles the errors gracefully and shows the user-friendly messages to the end user.
- **Stress testing:** Test the software by subjecting it to stressed conditions. Usually testers tend to test in non-stressed conditions. For Examples:- Try to save a huge file on low memory disk, or try to print a document with 2500 pages in it, try to save a form with maximum data in all the fields etc.
- **User Scenarios:** User Scenarios requires feature to work. User scenarios can be executed earlier after functionality testing to see if the feature interacts well with other product features and generates correct results as expected.

IV. CONCLUSIONS

Software testing is very important element of software development life cycle (SDLC). For a software product, usually around 50% of money and resources go under software testing. Software testing can furnish excellent results if done properly and effectively. Test cases are most important elements of software testing. If we design test cases in better manner and in properly way using test cases designing techniques then we can result with better software testing of a software product. By doing this we will get the result as we want in the requirement specified in the software requirement specification (SRS). This paper mainly deals with testing techniques available for designing of test cases. In the future, we can implement these test cases design techniques for real-time scenarios projects.

REFERENCES

[1] Antonia Bertolina, "*Software Testing Research and Practice*", Proceedings of the Abstract state machines 10th international conference on advances in theory and practice.

[2] Mohd. Ehmer Khan, "*Different forms of software testing techniques for finding errors*", IJCSI International Journal
Of Computer Science Issues, Vol. 7, Issue 3, No. 1, May 2010.

[3] Sahil Batra, Dr. Rahul Rishi, "*Improving Quality Using testing strategies*", Journal of Global Research In Computer Science, Volume 2, No. 6, June 2011.

[4] Abhijit A. Sawant, Pranit H. Bari and P.M. Chawan, "*Software testing techniques and Strategies*", International Journal of Engineering Research and Applications (IJERA), pp.980-986, Vol. 2, Issue 3, May-June 2012.

[5] Dolores R. Wallace, Laura M. Ippolito, Barbara B. Cuthill, "*Reference Information for the software Verification and Validation Process*", DIANE Publishing, 1996.

[6] Drake, T.(1996), "*Measuring Software quality: A case study*". IEEE Computer, 29(11), 78-87.

[7] Sheetal Thakre, Savita Chavan, Prof. P.M. Chawan, "*Software Testing Strategies and Techniques*", International Journal of Emerging Technology and Advanced Engineering, pp.567-569, Vol. 2, Issue 4, April 2012.

[8] Edward L. Jones "Grading student programs- a software testing", Proceedings of the fourteenth annual Consortium for computing Sciences in Colleges, 2000.

[9] Ian Somerville, "*Software Engineering*", Addison-Wesley, 2001.

[10] Rajib Mall, "*Fundamentals of software engineering*", The prentice hall of India, 3rd Edition, 2011.

[11] Peter Sestoft, "*Systematic software testing*", Version 2, 008-02-25.

[12] Gaurav Saini, Kestina Rai, "*An Analysis on Objectives, Importance and Types of Software Testing*", International Journal of Computer Science and Mobile Computing (IJCSMC), Vol. 2, Issue 9, September 2013, pp.18-23.

[13] Rajat kumar Bal, "*Software Testing*".

