Survey- Big Data Using Software Quality Testing with Techniques & Methodology

Priya shah
Assistant professor
Department of computer application
Patel Group of Institutions, GTU Mehsana (Gujarat, India)

Abstract

Software testing is a process of verifying and validating that a software application or program works as per the user's expectations. It is used to find out the important defects, flaws, or errors in the application code. In this paper we have developed a tool to generate different test cases automatically. In Model Based Testing (MBT), test cases are generated automatically from a partial representation of expected behavior of the System under Test (SUT) (i.e., the model). For most industrial systems, it is impossible to generate all the possible test cases from the model. The test engineer recourse to generation algorithms that maximize a given coverage criterion, a metric indicating the percentage of possible behaviors of the SUT covered by the test cases. Our previous work redefined classical Transition Systems (TSs) criteria for SPLs, using Featured Transition Systems (FTSs), a mathematical structure to compactly represent the behavior of a SPL, as model for test case generation. In this paper, we provide one all-states coverage driven generation algorithm and discuss its scalability and efficiency with respect to random generation. All-states and random generation are compared on fault-seeded FTSs.

Keywords—*Software Testing, Verification, Validation, Test Cases, Boundary Value Analysis,Equivalence Class Partition, Agile Method.*

Introduction

With the growing complexity of today's software applications injunction with the increasing competitive pressure has pushed the quality assurance of developed software towards new heights. Software testing is an inevitable part of the Software Development Lifecycle, and keeping in line with its criticality in the pre- and post-development process makes it something that should be catered with enhanced and efficient methodologies and techniques. This paper aims to discuss the existing as well as improved testing techniques for the better-quality assurance purposes.

In the place of traditional principle of project management, a strategic management philosophy is emerging fast in which writing better test cases also receive the widespread attention of all those interested in software project management and software testing. In the current scenario managing software is an important task in an IT industry. Not only managing IT project, but also it is needing to develop quality software product for the customer. It includes the number of tasks and phases of the software project development. Testing is one of the phases, which is most important in project management. In software testing writing test cases is very important. So, it is necessary to study how to write better test cases. This paper describes how to avoid loses that is inevitable with poor test cases.. Case study tries to give an insight about how to use test cases to improve testability and productivity, how to solve familiar challenges to test case quality and how to protect test case assets, which can be in practice in the software industry [1].

A test case is a set of conditions under which a tester will determine that an application, functionality, or software is working as expected. The test case has preconditions, steps, and an expected result. Test cases focus on testing small pieces of functionality. Testing is defined as a process of evaluation that either the specific system meets its originally specified requirements or not. It is mainly a process encompassing validation and verification process that whether the developed system meets the requirements defined by user. Therefore, this activity results in a difference

between actual and expected result. Software Testing refers to finding bugs, errors or missing requirements in the developed system or software.

Case Study of Test Cases

Test Case is a commonly used term for a specific test. This is usually the smallest unit of testing.
A Test Case will consist of information such as requirements testing, test steps, verification steps, prerequisites, outputs, test environment, etc.
A test case is a detailed procedure that fully tests a feature or an aspect of a feature.
Organizations take a variety of approaches to documenting test cases; these range from developing detailed, recipe-like steps to writing general descriptions. In detailed test cases, the steps describe exactly how to perform the test. In descriptive test cases, the tester decides at the time of the test how to perform the test and what data to use.
This is particularly important if you plan to compare the results of tests over time, such as when you are optimizing configurations. Detailed test cases are more time-consuming to develop and maintain. On the other hand, test cases that are open to interpretation are not repeatable and can require debugging, consuming time that would be better spent on testing.
When planning your tests, remember that it is not feasible to test everything. Instead of trying to test every combination, prioritize your testing so that you perform the most important tests those that focus on areas that present the greatest risk or have the greatest probability of occurring first. Once the Test Lead prepared the Test Plan, the role of individual testers will start from the preparation of Test Cases for each level in the Software Testing like Unit Testing, Integration Testing,
The bolded words should be replaced with the actual Project Name, Version Number and Release Date. We have company emblem and we will fill the details like Project ID, Project Name, Author of Test Cases, Version Number, Date of Creation and Date of Release in this Template. And we will maintain the fields Test Case ID,

Test Case ID: To Design the Test Case ID also we are following a standard: If a test case belongs to application not specifically related to a Module then we will start them as TC001, if we are expecting more than one expected result for the same test case then we will name it as TC001.1. If a test case is related to Module then we will name it as M01TC001, and if a module is having a sub-module then we name that as M01SM01TC001. So that we can easily identify to which Module and which sub-module it belongs to. And one more advantage of this convention is we can easily add new test cases without changing all Test Case Number so it is limited to that module only
Requirement Number: It gives the reference of Requirement Number in SRS/FRD for Test Case. For Test Case we will specify to which Requirement it belongs to. The advantage of maintaining this one
here in Test Case Document is in future if a requirement will get change then we can easily estimate how many test cases will affect if we change the corresponding Requirement.
Version Number: Under this column we will specify the Version Number, in which that test case was introduced. So that we can identify finally how many Test Cases are there for each Version.
Type of Test Case: It provides the List of different type of Test Cases like GUI, Functionality, Regression, Security, System, User Acceptance, Load, Performance etc
Test Case Name: This gives more specific name like Button or text box name, for which that Test Case belongs to. I mean to say we will specify the Object name for which it belongs to. For egg., OK button, Login form.
Action: This is very important part in Test Case because it gives the clear picture what you are doing on the specific object. We can say the navigation for this Test Case. Based the steps we have written here we will perform the operations on the actual application.
Expected Result: This is the result of the above action. It specifies what the specification or user expects from that action. It should be clear and for each expectation we will sub-divide that Test Case..

Actual: We will test the actual application against each Test Case and if it matches the Expected result then we will say it as "As Expected" else we will write the what happened after doing those action.
Status: It simply indicates Pass or Fail status of that Test Case.

Format of Standard Test Cases

Optionally you can have the following fields depending on the project requirements

- **Link / Defect ID**: Include the link for Defect or determine the Defect number if test status is fail
- **Keywords / Test Type**: To determine tests based on test types this field can be used. Egg: Usability, functional, business rules, etc.
- **Requirements**: Requirements for which this test case is being written
- **References / Attachments**: It is useful for complicated test scenarios, give the actual path of the document or diagram
- **Automation (Yes/No)**: To track automation status when test cases are automated
- **Custom Fields**: Fields your project being tested due to client/project requirements.

Table of Test Cases.

Use Technics-

There are using two technics: -

1) Boundary Values Analysis.
2) Equivalence Class Partitioning.

Boundary Values Analysis

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

The basic idea in boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum

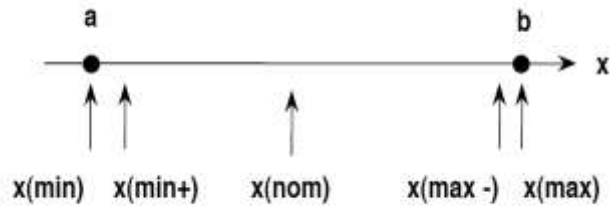

Fig.1. Figure of BVA.

In Boundary Testing, Equivalence Class Partitioning plays a good role

Equivalent Class Partitioning

Equivalent Class Partitioning is a black box technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system, etc. In this technique, you divide the set of test condition into a partition that can be considered the same.

Case 1:

Equivalence and Boundary Value Let's consider the behaviour of tickets in the Flight reservation application, while booking a new flight.

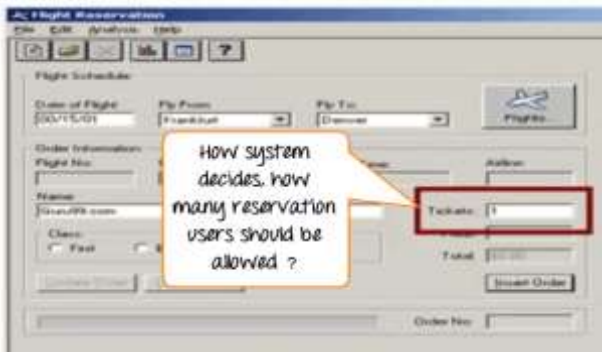

Fig. 2. Figure of ECP & BVA.

Ticket values 1 to 10 are considered valid & ticket is booked. While value 11 to 99 are considered invalid for reservation and error message will appear, "Only ten tickets may be ordered at one time."

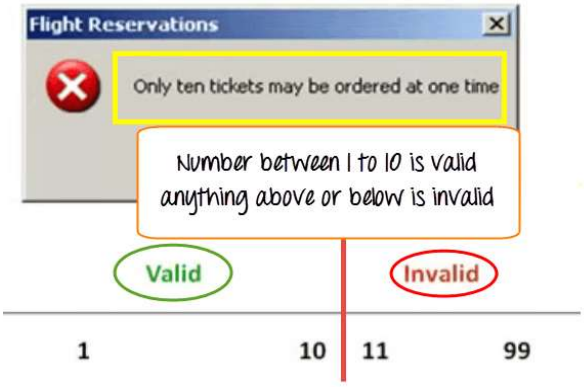

Fig. 3 Figure of ECP.

Here is the test condition

1. Any Number greater than 10 entered in the reservation column (let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behaviour can be considered the same.

Fig. 4 Figure of ECP.

Then we pick only one value from each partition for testing. The hypothesis behind this technique is that if one condition/value in a partition passes all others will also pass. Likewise, if one condition in a partition fails, all other conditions in that partition will fail. [2]

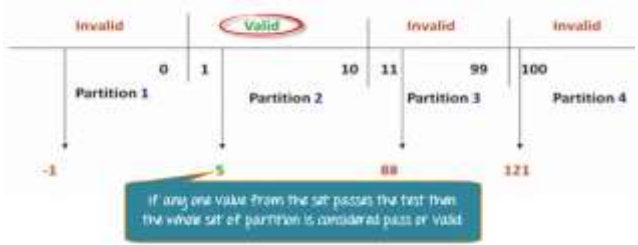

Fig. 5 Figure of BVA & ECP.

Boundary Value Analysis- in Boundary Value Analysis, you test boundaries between equivalence partitions.

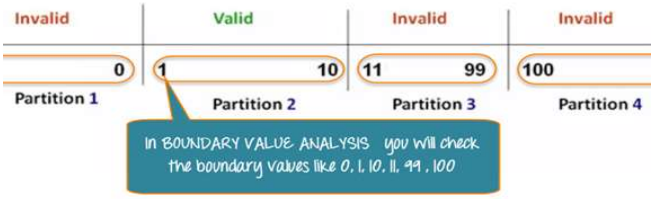

Fig. 6 Figure of BVA.

In our earlier example instead of checking, one value for each partitions you will check the values at the partitions like 0, 1, 10, 11 and so on. As you may observe, you test values at both valid and invalid boundaries.

Case 2:
Equivalence and Boundary Value
Suppose a password field accepts minimum 6 characters and maximum 10 characters

That means results for values in partitions 0-5, 6-10, 11-14 should be equivalent

Table of Test Cases Scenario.

Test Scenario #	**Test Scenario Description**	**Expected Outcome**
1	Enter 0 to 5 characters in password field	System should not accept
2	Enter 6 to 10 characters in password field	System should accept
3	Enter 11 to 14 character in password field	System should not accept

Case 3:
Input Box should accept the Number 1 to 10
Here we will see the Boundary Value Test Cases.

Table of Test Design Scenario.

Test Scenario Description	**Expected Outcome**
Boundary Value = 0	System should NOT accept
Boundary Value = 1	System should accept
Boundary Value = 2	System should accept
Boundary Value = 9	System should accept
Boundary Value = 10	System should accept

Boundary Value = 11	System should NOT accept

Why Equivalence & Boundary Analysis Testing

1. This testing is used to reduce very large number of test cases to manageable chunks.
2. Very clear guidelines on determining test cases without compromising on the effectiveness of testing.
3. Appropriate for calculation-intensive applications with substantial number of variables/inputs

Summary:
Boundary Analysis testing is used when practically it is impossible to test large pool of test cases individually Two techniques - Equivalence Partitioning & Boundary Value Analysis testing techniques is used
In Equivalence Partitioning, first you divide a set of test condition into a partition that can be considered.
In Boundary Value Analysis you then test boundaries between equivalence partitions Appropriate for calculation-intensive applications with variables that represent physical quantities

Method-

Agile Model
Agile methods took over the traditional methods, to overcome the rigidity of the traditional model. Agile follows a dynamic approach to software development. It is an interactive and team based method that aims to deliver an application in a short span of time. [4]
In agile methodology, tasks are categorised into phases and are "time-boxed", that is, time frames are allotted to each task. Each time-boxed phrase is called a sprint. Each sprint has a defined duration of time, say, a week, few days or month.

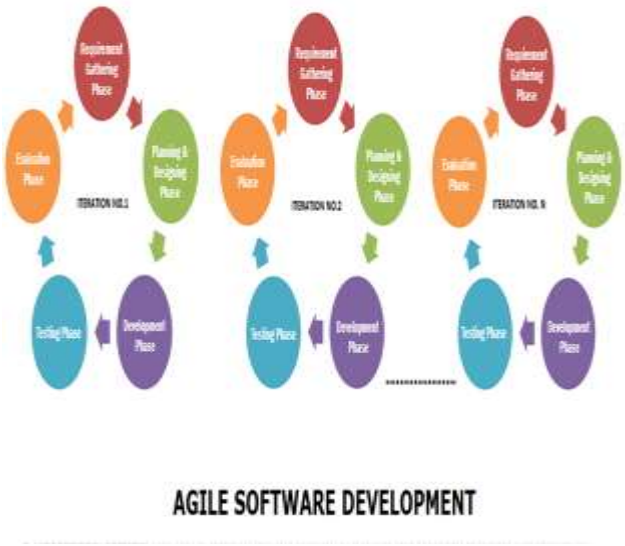

Fig. 7. Figure of agile Model.

Agile is based on empirical process, which provides a control mechanism based on a defined set of methods. Empirical method is meant for those processes that are not very well defined, unpredictable or unrepeatable. Agile technique implements control through frequent inspection and adaptation.
It has brief iterative life cycles, which reflects periodic changes, and thus integrating slight change cycle to the overall system development process.
Involves communication with customers consistently, taking their feedback as input, during the different iterative cycles. [3]

Agile Vs Traditional

Table--Comparison between two models.

S.No.	Traditional Model	Agile Model
1.	Follows a top down approach, and making changes is not easy as finishing one phase leads to another	Team conducts experiments on various techniques and gradually arrives at the best possible solution
2.	It has a leadership style of working	In agile, there is free flow of communication, anyone can present their ideas within the team
3.	Pre-planning is done to carry out the various phases	This is more flexible as compared to traditional model, as it can change its work flow based on any new request for modifications
4.	Customer is involved only in the initial phases of requirements gathering	Customer involvement is crucial for this model to prove its mettle
5.	The project plan is prepared before commencing the process of system development	Project work is delivered to the client in small amount, that is, as and when one module is prepared, a demonstration is given to the

		client, to confirm the work progress in a right direction
6.	The ownership lies on the Project manager	It has the concept of shared ownership, i.e, every team member is equally responsible for their individual contribution
7.	Believes in one-time delivery of the product	Relies on incremental delivery of the product

Both agile and traditional models are essential for an efficient software development process.
Agile Model does overcome few deficiencies that the traditional model imbibes, but at the same time each model's pros and cons must be weighed before reaching a consensus.

Conclusion:

There are substantial numbers of people, which are being victimised by the myths, associated with software testing, and consider testing, inferior to the development process. However, the truth is very much different. Testing phase is as important as the development phase. Similar, to the development process, testing is a wider concept that encompasses numerous types of activities.

References

[1]. Neha Kumawat, Yashika Sharma, Urmila Pilania, Comparative Study between Equivalence Class Partitioning and Boundary Value Analysis Testing Methods, © March 2016 | IJIRT | Volume 2 Issue 10 | ISSN: 2349-6002.

[2]. Patrcia Frankovaa, Martina Drahosovab, Peter Balcoa, Agile project management approach and its use in big data management,7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)

[3]. Harmeet Kaur, Shahanawaj Ahamad, Gurvinder N. Verma, Identification & Analysis of Parameters for Program Quality Improvement: A Reengineering Perspective, Computer Engineering and Intelligent Systems Vol.4, No.5, 2013.

[4]. Rashmi Popli, Design of Sprint Point Based Estimation Techniques for Agile Software, YMCA University of Science and Technology

[5]. Shishank Gupta, Parametric Test Optimization, Infosys.
[6]. Reference Book from Guru.99.