Basic Concepts of Software Testing and Software Testing Strategies

Shubham Singhal
B.Tech(CSE), Galgotias University, Greater Noida, Uttar Pradesh, India

***Abstract*- Software Testing plays an important role to achieve and improve the software product quality. On one side, during the development of the software we improve the quality of the product as we repeat a test-find defects-fix cycle. On the other side, before the release of a product in the market we perform system-tests to determine how accurate our system is performing. A software testing strategy is a process, for evaluating the functionality and behavior of a software product with the intent to find that the software developed met the specified requirements or not. A software testing strategy is an outline describing the software development cycle testing approach. These are basically the QA strategies describe ways of mitigating product risks of stakeholders in the test level, the kind of testing to be performed and which entry and exit criteria would apply.**

Index terms- Software Testing, test-find defects-fix cycle, system-tests, specified requirements, software testing strategy, QA strategy

I. INTRODUCTION

Software testing is a verification process conducted for software quality assessment and improvement. Generally speaking, the activities for software quality assessment can be grouped into two broad categories, namely, static analysis and dynamic analysis.

A. Static Analysis

It is based on the examination of various documents, namely requirements documents, software models, design documents, and source code. Traditional static analysis includes code review, inspection, walk-through, the analysis of algorithm, and the proof of correctness. It does not involve the actual execution of the code which is under development. Instead, it examines the code and reasons over all possible behaviors that might arise during run time. Compiler optimizations are standard static analysis.

B. Dynamic Analysis

The Dynamic analysis of a software system involves actual program execution in order to uncover possible program failures. The performance properties of the program are also observed. Software Programs are executed and evaluated with both typical and carefully chosen input values. Many times, the input set of a program can be impractically large. However, for practical situations, a finite subset of the input set can be selected. Therefore, in software testing, we observe some representative program behaviors and reach the conclusion about the quality of the system. Careful selection of finite test set is crucial to reaching a reliable conclusion.

II. VERIFICATION AND VALIDATION

Two similar concepts related to software testing frequently used are verification and validation. Both verification and validation concepts are abstract in nature, and each can be realized by a set of concrete, executable activities. The two concepts are explained as follows:

A. Verification

The verification activity helps us in evaluating the software system by determining whether the product of a given development phase satisfies the expectations and requirements specified before the start of that phase. One may note that a product can be an intermediate product, such as requirement specification, design specification, coding details, user manual, or even the final product. These activities that check the correctness of a development phase are called verification activities.

B. Validation

The validation activity helps us in confirming that a product meets its intended use. Validation activities aim at confirming that a product meets its customer's

expectations. In other words, validation activities focus on the final product, which is tested extensively from the customer point of view. Thus validation establishes whether the product meets overall expectations of the users.

III. FAILURE, ERROR, FAULT, AND DEFECT

In literature on software testing, one can find references to the terms failure, error, fault, and defect. Although their meanings are related, but there are some important distinctions between these four concepts. In the following, we present first three terms as they are understood in fault-tolerant computing community:

A. Failure

A failure is said to occur whenever external behavior of a system does not conform to that prescribed in the system specification.

B. Error

An error is a state of the system. In absence of any action by system for the correction, an error state could lead to a failure which would not be attributed to any event subsequent to the error.

C. Fault

A fault is basically the adjudged cause of an error.

IV. OBJECTIVES OF TESTING

The stakeholders in a testing process are the programmers, the test engineers, the project managers, and the customers. Different stakeholders view a test process from the different perspectives as explained below:

A. It does work

While implementing a program code, the programmer may want to test whether or not the coding unit works in normal circumstances. Programmer gets much confidence if the unit works to his/her satisfaction. This same idea applies to an entire system as well—once a system has been integrated, the developers or programmers may want to test whether or not the system performs the basic functions.

B. It does not work

Once the programmer (or the developer) is satisfied that a programming unit (or the system) works to a certain degree, more tests are conducted with the objective of finding faults in the unit (or the system). Here, idea is to try to make the unit (or the system) fail.

C. Reducing the risk of failure

Most of the complex software systems contain faults, which cause system to fail from time to time. This concept of "failing from time to time" gives rise to the notion of failure rate. As the faults are identified and fixed while performing more and more tests, failure rate of a system generally decreases. Thus, a higher level objective of performing the tests is to bring down the risk of failing to the acceptable level.

D. Reducing the cost of testing

The different types of costs associated with a test process include:

- cost of designing, maintaining, and executing the test cases,
- cost of analyzing the results of executing each test case,
- cost of documenting the test cases, and
- cost of actually executing system and documenting it.

Therefore, less the number of test cases designed, less will be the associated cost of testing.

V. PRINCIPLES OF TESTING

Software testing is a process of executing a program with the aim of finding the error. In order to make our software perform well it should be error free. If testing is done successfully it will remove all the errors from the software.

A. Testing shows presence of defects

The primary goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn't talk about the absence of defects. Software testing can ensure that defects are present but it cannot prove that software is defects free. Even multiple testing can never ensure that software is completely 100 percent bug-free. Testing can reduce the number of defects but not removes all defects.

B. Exhaustive testing is impossible

It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some of the test cases and assume that software is correct and it will produce the correct output in every test case. If the software will test each and every test case then it will take more cost, effort, etc. and which is impractical.

C. Early Testing

To uncover the defect in the software, early test activity shall be started. The defect detected in early phases of SDLC will very less expensive. For the better performance of software, software testing will start at initial phase i.e. testing will perform at the requirement analysis phase.

D. Defect clustering

In the software project, a small number of the module can contain most of the defects. The principle to software testing state that 80% of software defect comes from 20% of modules.

E. Pesticide paradox

Repeating same test cases again and again will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

F. Testing is context dependent

The approach of testing depends on context of software developed. Different types of software need to perform different types of testing. For example, the testing of the e-commerce site is different from the testing of the Android application.

G. Absence-of-errors fallacy

If the built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it also mandatory to fulfill all the customer requirements.

VI. LEVELS OF SOFTWARE TESTING

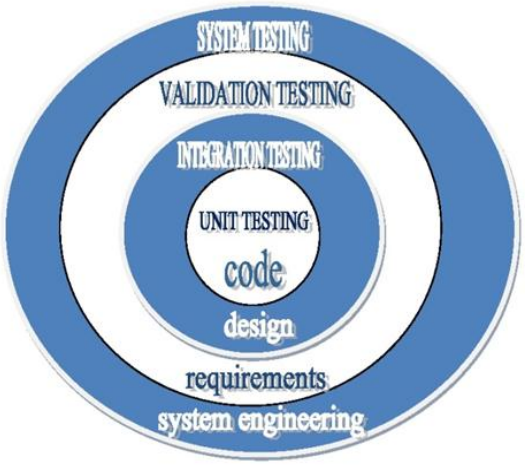

fig: Levels of Software Testing

As we know that the during the software development, the software is built step by step that is starting from gathering the requirements and expectations of the customer to the finally by coding it to keep it properly. So, according to the above diagram the software development takes place by moving INWARDS in the spiral. Now, the testing of the software is done in the reverse order by moving OUTWARDS in the spiral. To ensure the software is as per the customer expectations and to test it properly there are four basic levels of testing performed on it, that are defined below:

A. Unit Testing

In the unit testing, each module of the software is tested individually to ensure that the every module produces its specific functionality based on the implementation. In this level of testing the modules are tested independently so that one module will not affect the functionality of another module. Unit's internal code/structure, path testing ensures complete coverage of unit. That is, it basically deals with the coding part of the software.

B. Integration testing

In this level of software testing, the components are integrated step by step to form the complete software and the integrated module is tested to ensure that the units which were previously working flawlessly performs appropriately upon integration also. The focus of the integration testing is on the software architecture and the construction of the software that is on the design of the software.

C. Validation Testing

In the validation testing, the software is tested against the SRS (System Requirement and Specification) to make sure that the developed software fulfills the requirements and expectations as specified by the customer. The task is to go through with the behavior, performance requirements and all kinds of validation criteria.

D. System Testing

This is the last level of testing conducted on the complete integrated system, done after the unit, integration and validation testing. It evaluates the system compliances with requirements. It brings out the defects that are not directly accessible to a module/interface but are based on issues that are related to design and architecture of the whole product. It basically deals with system engineering of the software product.

VII. SOFTWARE TESTING STRATEGIES

A software testing strategy is like a blueprint which describes the software development cycle testing approach. It is made to inform testers, managers and developers on some major issues of the testing process. This includes testing objective, total time and resources needed for a project, methods of testing new functionalities and the testing environment.

Software testing strategies describe how to mitigate product risks of stakeholders at the test level, which kinds of testing are to be done and which entry and exit criteria will apply. They're made based on development design documents.

VIII. FACTORS TO CONSIDER IN CHOOSING SOFTWARE TESTING STRATEGIES

A. Risks

The Risk management is paramount during testing, thus consider the risks and the risk level. For an application that is well-established that's slowly evolving, regression is a critical risk. That is why regression-averse strategies make a lot of sense. For a new application, a risk analysis could reveal various risks if choosing a risk-based analytical strategy.

B. Objectives

Testing should satisfy the requirements and needs of stakeholders to succeed. If objective is to look for as many defects as possible with less time and effort invested, a dynamic strategy makes sense.

C. Skills

The skills that testers possess and lack must be taken into consideration, since strategies should not only be chosen but executed as well. A standard compliant strategy is a good option when lacking skills and time in the team to create an approach.

D. Product

Some products such as project development software and weapons systems tend to have requirements that are well-specified. This could lead to synergy with the analytical strategy that is requirements-based.

E. Business

Business considerations and strategy are often important. If using a legacy system as a model for the new one, one could use a model-based strategy.

F. Regulations

At some instances, one may not only have to satisfy stakeholders, but regulators as well. In this case, one may require a methodical strategy which satisfies these regulators.

You must select testing strategies with an eye towards the factors mentioned earlier, the schedule, budget, and feature constraints of the project and the realities of the organization and its politics.

VIII. STRATEGIES IN SOFTWARE TESTING

A good software testing or QA strategy requires tests at all technology stack levels to ensure that every part, as well as the entire system, works correctly.

A. Leave time for fixing

Setting aside time for testing is of no use if there is no time set aside for fixing. Once problems are discovered, developers required time to fix them and the company needs time to retest the fixes as well. With a time and plan for both, then testing is not very beneficial.

B. Discourage passing the buck

The same way that testers could fall short in their reports, developers could also fall short in their effort to comprehend the reports. One way of minimizing back and forth conversations between developers and testers is having a culture that will encourage them to hop on the phone or have desk-side chat to get to the bottom of things. Testing and fixing are all about collaboration. Although it is important that developers should not waste time on a wild goose chase, it is equally important that bugs are not just shuffled back and forth.

C. Manual testing has to be exploratory

A lot of teams generally prefer to script manual testing so testers follow a set of steps and work their way through a set of tasks that are predefined for software testing. This misses the point of manual testing. If anything could be written down or scripted in exact terms, it could be automated and belongs in the automated test suite. The real-world use of the software will not be scripted, thus testers must be free to probe and break things without a script.

D. Encourage clarity

Reporting bugs and asking for more information could create unnecessary overhead costs. A good bug report could save time through avoiding miscommunication or a need for more communication. In the same way, a bad bug report could lead to a fast dismissal by a developer. These could create problems. Anyone reporting bugs should make it a point to create bug reports that are informative. However, it is also integral for the developer to out of the way to effectively communicate as well.

E. Test often

Same as all the other forms of testing, manual testing will work best when it occurs often throughout the development project, in general, weekly or bi-weekly. This helps in preventing huge backlogs of problems from building up and crushing morale. Frequent testing is considered the best approach.

Testing and fixing the software product could be tricky, subtle and political even. Nevertheless, as long as one is able to anticipate and recognize common issues, things could be kept running smoothly.

IX. CONCLUSION

Testing is a crucial activity to be performed during the software development. Software testing can give confidence in the quality of the software if it finds few or no defects. If defects are found, the quality increases when those defects are fixed. So here we learned about all the basic concepts of the software testing activity. Apart from that we also discussed about what all the software testing strategies we need to perform to test out the software product. This guided us to understand the software testing process along with their strategies in detail.

APPENDIX

A. Positive Testing

Operate applications or software as it should be operated. Use proper variety of test data, including data values at boundaries to test if it fails. Verify actual test results with the expected and see

- Does it behave normally?
- Are results correct?
- Does the software function correctly?

B. Negative Testing

Test for abnormal operations. Test with illegal/abnormal test data. Intentionally attempt to make things go wrong and to discover/detect and see

- Does the system fail/crash?
- Does the program do what it should not?
- Does it fail to do what it should?

ACKNOWLEDGEMENT

First & foremost, thanks to the God for his blessings throughout my research work to complete the research successfully.

I would like to express my deep and sincere gratitude to my parents for their love, prayers, care and sacrifices for educating and preparing me for my future. I would also like to express my gratefulness to all my friends and family for their constant encouragement.

Finally, my special thanks to all the people who have supported and encouraged me to complete the research work directly or indirectly.

REFERENCES

[1] Kshirasagar Naik and Priyadarshi Tripathy, Software Testing and Quality Assurance: Theory and Practice, Wiley Publications
[2] S. Desikan and G. Ramesh, Software Testing: Principles and Practices, Pearson Education
[3] Aditya P. Mathur, Fundamental of Software Testing, Pearson Education
[4] K. K. Aggarwal and Yogesh Singh, Software Engineering, New Age International Publication.