Principles and Goals of Software Testing

Dr Leelavathi Rajamanickam

Senior Lecturer, School of Information Technology, SEGI University, Malaysia

Abstract— *Software testing is an activity which is aimed for evaluating quality of a program and also for improving it, by identifying defects and problems. Software testing strives for achieving its goal (both implicit and explicit) but it has certain limitations, still testing can be done more effectively if certain established principles are to be followed. In spite of having limitations, software testing continues to dominate other verification techniques like static analysis, model checking and proofs. So it is indispensable to understand the goals, principles and limitations of software testing so that the effectiveness of software testing could be maximized.*

Keywords— Software Testing, Testing tools, Testing principles, Testing Limitations

I. INTRODUCTION

Software testing is about testing a feature with varying test data to get a result and then comparing the actual result with expected result, it is not merely finding defects or bugs in the software; it is the completely dedicated discipline of evaluating the quality of the software [1]. Software testing is a process of verifying and validating that a software application or program meets the business and technical requirements that guided its design and development and works as expected and also identifies important errors or flaws categorized as per the severity level in the application that must be fixed [6]. Software testing is a phase of SDLC that entails much effort, time and cost. Often, testing phase is the single largest contributor towards the whole development time. Testing can not only uncover bugs in the program, but also flaws in design of the software [2]. Software testing is also used to test the software for other software quality factors like reliability, usability, integrity, security, capability, efficiency, portability, maintainability, compatibility etc. Testing approach differs for different software's, level of testing and purpose of testing.

Software testing should be performed efficiently and effectively, within the budgetary and scheduling limits. Following established principles can make testing easier and more effective, and can also ensure that testing goals are achieved to its maximum. They also ensure that a process is repeatable. Software testing is a very important quality filter and needs to be planned taking into account its goals and principles.

II. TESTING GOALS

A goal is a projected state of affairs that a person or system plans or intends to achieve. A goal has to be accomplishable and measurable. It is good if all goals are interrelated. In testing we can describe goals as intended outputs of the software testing process. Software testing has following goals:

2.1 Verification and Validation

It would not be right to say that testing is done only to find faults. Faults will be found by everybody using the software.

Testing is a quality control measure used to verify that a product works as desired [8]. Software testing provides a status report of the actual product in comparison to product requirements (written and implicit). Testing process has to verify and validate whether the software fulfills conditions laid down for its release/use [6]. Testing should reveal as many errors as possible in the software under test, check whether it meets its requirements and also bring it to an acceptable level of quality.

2.2 Priority Coverage

Exhaustive testing is impossible [7]. We should perform tests efficiently and effectively, within budgetary and scheduling limitations. Therefore testing needs to assign effort reasonably and prioritize thoroughly. Generally every feature should be tested at least with one valid input case. We can also test input permutations, invalid input, and non-functional requirements depending upon the operational profile of software. Highly present and frequent use scenarios should have more coverage than infrequently encountered and insignificant scenarios. A study by on 25 million lines of code also revealed that 70-80% of problems were due to 10-15% of modules, 90% of all defects were in modules containing 13% of the code, 95% of serious defects were from just 2.5% of the code. Pareto principle also states that 80 percent of all software defects uncovered during testing will likely be traceable to 20 percent of all program components. The problem, of course, is to isolate these suspect components and to thoroughly test them. Overall we target a wide breadth of coverage with depth in high use areas and as time and budget permits.

2.3 Balanced

Testing process must balance the written requirements, real-world technical limitations, and user expectations.

The testing process and its results must be repeatable and independent of the tester, i.e., consistent and unbiased. Apart from the process being employed in development there will be a lot unwritten or implicit requirements. While testing, the software testing team should keep all such requirements in mind. They must also realize that we are part of development team, not the users of the software. Testers personal views are but one of many considerations. Bias in a tester invariably leads to a bias in coverage. The end user's viewpoint is obviously vital to the success of the software, but it is not all that matters as all needs cannot be fulfilled because of technical, budgetary or scheduling limitations. Every defect/shortcoming has to be prioritized with respect to their time and technical constraints.

2.4 Traceable

Documenting both the successes and failures helps in easing the process of testing. What was tested, and how it was tested, are needed as part of an ongoing testing process. Such things serve as a means to eliminate duplicate testing effort [8]. Test plans should be clear enough to be re-read and comprehended. We should agree on the common established documentation methods to avoid the chaos and to make documentation more useful in error prevention.

2.5 Deterministic

Problem detection should not be random in testing. We should know what we are doing, what are we targeting, what will be the possible outcome. Coverage criteria should expose all defects of a decided nature and priority. Also, afterward surfacing errors should be categorized as to which section in the coverage it would have occurred, and can thus present a definite cost in detecting such defects in future testing. Having clean insight into the process allows us to better estimate costs and to better direct the overall development.

III. TESTING PRINCIPLES

A principle is an accepted rule or method for application in action that has to be, or can be desirably followed. Testing Principles offer general guidelines common for all testing which assists us in performing testing effectively and efficiently. Principles for software testing are:

3.1 Test a program to try to make it fail

Testing is the process of executing a program with the intent of finding errors [7]. Our objective should be to demonstrate that a program has errors, and then only true value of testing can be accomplished. We should expose failures (as many as possible) to make testing process more effective.

3.2 Start testing early

Testing a program involves providing the program with set of test inputs (or test cases) and observing if the program behaves expected. If the program fails to behave as expected, then the input data and the conditions under which the failure occurs are noted for later debugging and error correction [2]. If you want to find errors, start as early as possible. This helps in fixing enormous errors in early stages of development, reduces the rework of finding the errors in the initial stages. Fixing errors at early phases cost less as compared to later phases. For example, if a problem in the requirements is found after releasing the product, then it would cost 10–100 times more to correct than if it had already been found by the requirements review. Fig. 1 depicts the increase in cost of fixing bugs detected/fixed in later phases.

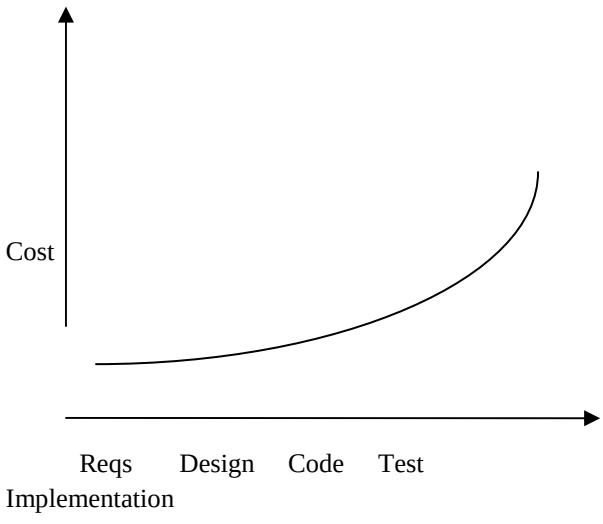

Fig. 1: Cost of the bugs

3.3 Testing is context dependent

Testing is done differently in different contexts. Testing should be appropriate and different for different points of time. For example, a safety-critical software is tested differently from an e-commerce site. Even a system developed using the waterfall approach is tested significantly differently than those systems developed using agile development approach. Even the objectives of testing differ at different point in software development cycle. For example, the objective of unit and integration testing is to ensure that code implemented the design properly. In system testing the objective is to ensure the system does what customer wants it to do [13]. Type of testing approach that will be used depends on a number of factors, including the type of system, regulatory standards, user requirements, level and type of risk, test objective documentation available, knowledge of the testers, time and budget, development life cycle.

3.4 Define Test Plan

Test Plan usually describes test scope, test objectives, test strategy, test environment, deliverables of the test, risks

and mitigation, schedule, levels of testing to be applied, methods, techniques and tools to be used. Test plan should efficiently meet the needs of an organization and clients as well. The testing is conducted in view of a specific purpose (test objective) which should be stated in measurable terms, for example test effectiveness, coverage criteria. Although the prime objective of testing is to find errors, a good testing strategy also assesses other quality characteristics such as portability, maintainability and usability.

3.5 Design Effective Test cases

Complete and precise requirements are crucial for effective testing. User Requirements should be well known before test case design. Testing should be performed against those user requirements. The test case scenarios shall be written and scripted before testing begins. If you do not understand the user requirements and architecture of the product you are testing, then you will not be able to design test cases which will reveal more errors in short amount of time. A test case must consist of a description of the input data to the program and a precise description to the correct output of the program for that set of input data. A necessary part of test documentation is the specification of expected results, even if providing such results is impractical [7]. These must be specified in a way that is measurable so that testing results are unambiguous.

3.6 Test for valid as well as invalid conditions

In addition to valid inputs, we should also test system for invalid and unexpected inputs/conditions. Many errors are discovered when a program under test is used in some new and unexpected way and invalid input conditions seem to have higher error detection yield than do test cases for valid input conditions [9]. Choose test inputs that possibly will uncover maximum faults by triggering failures.

3.7 Review Test cases regularly

Repeating same test cases over and over again eventually will no longer find any new errors. Therefore the test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to potentially find more defects. We should target and test susceptible areas. Exploratory Testing can prove very useful. Exploratory testing is any testing to the extent that the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests[7].

3.8 Testing must be done by different persons at different levels

Different purposes are addressed at the different levels of testing. Factors which decide who will perform testing include the size and context of the system, the risks, the development methodology used, the skill and experience of the developers.

Testing of individual program components is usually the responsibility of the component developer (except sometimes for critical systems); Tests at this level are derived from the developer's experience. Testing at system/sub-system level should be performed by the independent persons/team. Tests at this level are based on a system specification. Development staff shall be available to assist testers. Acceptance Testing is usually performed by end user or customer. Release Testing is performed by Quality Manager. Fig. 2 shows persons involved at different levels of software testing.

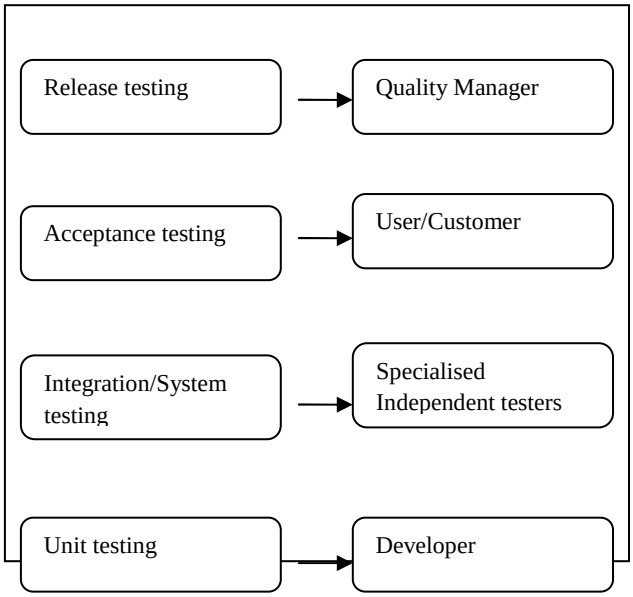

*Fig. 2***:** *Software Testing Levels***.**

3.9 Test a program innovatively

Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. It is impossible to test a program sufficiently to guarantee the absence of all errors [9]. Instead of exhaustive testing, we use risks and priorities to focus testing efforts more on suspected components as compared to less suspected and infrequently encountered components.

3.10 Use both Static and Dynamic testing

Static testing is good at depth; it reveals developers understanding of the problem domain and data structure.

Dynamic testing is good at breadth; it tries many values, including extremes that humans might miss. To eliminate as many errors as possible, both static and dynamic testing should be used [10].

3.11 Defect clustering

Errors tend to come in clusters. The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section [9], so additional testing efforts should be more focused on more error-prone sections until it is subjected to more rigorous testing.

3.12 Test Evaluation

We should have some criterion to decide whether a test is successful or not. If limited test cases are executed, the test oracle (human or mechanical agent which decides whether program behaved correctly on a given test [3]) can be tester himself/herself who inspects and decides the conditions that makes test run successful. When test cases are quite high in number, automated oracles must be implemented to determine the success or failure of tests without manual intervention. One good criterion for test case evaluation is test effectiveness (number of errors it uncovers in given amount of time).

3.13 Error Absence Myth

System that does not fulfill user requirements will not be usable even if it does not have any errors. Finding and fixing defects does not help if the system built does not fulfill the users' needs and expectations. In addition to positive software testing (which verify that system does what it should do), we should also perform negative software testing (which verify that system does not do what it should not do).

3.14 End of Testing

Software testing is an ongoing process, which is potentially endless but has to be stopped somewhere. Realistically, testing is a trade-off between budget, time and quality [11]. The effort spent on testing should be correlated with the consequences of possible program errors [11]. The possible factors for stopping testing are:

1. The risk in the software is under acceptable limit.
2. Coverage of code/functionality/requirements reaches a specified point.
3. Budgetary/scheduling limitations.

IV. CONCLUSION

Software testing is a vital element in the SDLC and can furnish excellent results if done properly and effectively. Unfortunately, Software testing is often less formal and rigorous than it should, and a main reason for that is because we have struggled to define best practices, methodologies, principles, standards for optimal software testing. To perform testing effectively and efficiently, everyone involved with testing should be familiar with basic software testing goals, principles and concepts. Already lot of work has been done in this field, and even continues today. Implementing testing principles in real world software development, to accomplish testing goals to maximum extent keeping in consideration the testing limitations will validate the research and also will pave a way for future research.

REFERENCES

[1] Leelavathi Rajamanickam, "Testing Tool for Object Oriented Software" International Journal of scientific research and management (IJSRM), **vol 2**. Issue 8 august 2014, pages: 1205-1208 (2014).

[2] Leelavathi Rajamanickam, "Software Testing, Analysis and Objectives" International Journal of Advanced Trends in Computer Science and Engineering, **Vol 3**, No.5, Pages: 01-04 (2014)

[3] Antonia Bertolina, "Software Testing Research and Practice", Proceedings of the abstract state machines 10th international conference on Advances in theory and practice, 1 -21, 2003.

[4] Drake, T. (1996) —"Measuring software quality: a case study." IEEE Computer, 29 (11), 78–87.

[5] [5] James Bach, —"Exploratory Testing Explained", vol 1.3 4/16/03.

[6] John E. Bentley, Wachovia Bank, Charlotte NC, —"Software Testing Fundamentals—Concepts, Roles, and Terminology", SUGI 30.

[7] Myers, Glenford J., —"The art of software testing", New York: Wiley, c1979. ISBN: 0471043281

[8] Nick Jenkins. —"A Software Testing Primer", 2008.

[9] Peter Sestoft, "Systematic software testing‖, Version 2, 2008-02-25.

[10] Programming Research Ltd, —"Static and Dynamic Testing Compared".

[11] Rajat Kumar Bal, "Software Testing".

[12] Salim Istaq et al., —"Debugging, Advanced Debugging and Runtime Analysis" —, (IJCSE) International Journal on Computer Science and Engineering‖ Vol. 2, No. 02, 2010, pages: 246-249.

[13] Shari Lawrence Pfleeger, —"Software Engineering, Theory and Practice", Pearson Education, 2001.