Biz and Bytes (Vol. 7. Issue: 1, 2016) E-ISSN: 0976 0458, Print ISSN: 2320 897X

Review on practices in Software Testing

Silky Sachar
Department of Computer Application,
CGC, Landran

Sonia Suneja
Department of Computer Application
CGC, Landran

Abstract
This paper strives to provide a complete view of the field of software testing. It attempts to consolidate all the relevant issues which can be faced in this field. It shows how a mix of theoretical and practical problems challenges the tester. How the test cases are deployed on modules. Formal and informal criteria for test selection are considered. Some other issues such as implementation in embedded system, outcome of test is acceptable or not and effectiveness of test. No separation between development and Testing is assumed.

Introduction
Software testing consists of executing the programs input in the form of value. This means that the behavior of the system mostly depends on the state of the system i.e. non- deterministic system can provide different output to the same input in the same state. The number of executions that can be implemented must be finite and cost effective. We always have constraints on schedules and resources and unlimited testing requirements. The next step is to identify the best criteria which is most suitable under the prevalent conditions in a complicated problem[1] (usually risk analysis techniques are used). The observed outcome is checked against the requirements provided by the user.

Selecting the Test Case
A "good" is the one which can detect a lot of Failures, it is therefore preferable to spent the budget (allotted for testing) to combine a different techniques [2] rather than on just one. It also affects other factors like reliability, usability,
Usually the test approaches are partitioned logically such that for every case within the same sub domain, the test proves to be a success or failure. The criterion for test selection depends on the kind of reference model [3] related to the program. It can be informal (dependent on expertise of tester) or formal (depends on formal specification or on the coverage of code).
The formal approach is usually preferred because it can be automated and evaluated. This can be achieved by modeling the program as a graph[4] and considering its entry exit points. However some issues arise. :-

- Some time the algorithm selects the path which cannot be executed
- To find an input which executes a particular path in graph (Undecidable problem)

Initial approaches manually derived boundary conditions by looking at input output relations in graphs [5]. To automate the derivation of test cases, early attempts included algebraic specifications [6], while a more recent perspective was found in [7]. The testing based on specification concentrate on testing from models developed in UML. Formal approaches are based on UML state charts [8].

Some other issues

There are some issues other than the selecting the test case which are very well known to the practitioner.

- The capability to implement the test in tight environment provided in embedded system
- To decide if the outcome of the test is acceptable or not. In case it is not then we need to evaluate the result of failure and eliminating its root cause.
- To find the effectiveness measures of the test to judge if the testing is sufficient.

Executing The Test

Once the test cases are derived ,we need to execute them. If the test cases were derived from code, then we can easily find out the inputs from entry exit points in gra**ph.** If the test cases were derived from specification then test cases depend on abstraction of specification.

Another important requirement is to bring the system in state in which test can be launched. Every test thus performed is supposed to leave the system in the same state [9]. To integrate test large complex systems, its required to define the tests as complete Input/output sequence. When testing results in failure then all the conditions need to be re-played, which can be synchronization of events or accessing the memory?

Evaluating the outcome

Testing is meaningful if the result can be evaluated accurately. The difficulties associated with it are referred in [10]. In case of less number of test cases which can be manually derived, the tester can inspect the conditions that cause the test to pass or fail and code them upon the test driver used.

In case the number of test cases exceeds the number which can he handled manually then this process is automated.

In case of regression testing where we need to compare the result of the test with the previous version executions, the tester(mechanical agent) needs to be derived from specification of expected behavior. That is, if we are automating the derivation of test cases, we can derive only necessary conditions for correctness.

Test Result Analysis

A test can be considered effective if the number of faults and errors are counted. If this number is high it implies that test was useful. A better approach here is to take the executed tests as sample of behavior of program, and make a prediction of what the future outcomes would be. This marks the base for reliability.

Software Reliability

Reliability is the measure of probability that a software will execute successfully in a given amount of time [11]. Reliability growth models are used after debugging is done, to ensure that once the program has been tested and failures have been determined then they need to be corrected and again submitted for testing until the desired reliability level is reached. Usually the reliability increases after software is repaired. It adopts statistical measures. If a set “s” of test cases is executed, “f” is the number of inputs that have failed, then reliability can be given by : R=1-f/s

Treating Development and Testing as a single unit

Despite of all the issues, the main challenge is to integrate testing and development. Hereby development and testing becomes two pass of a single unit. The main reason of considering testing as a second class activity is the separation between development and testing. Although testing is important but one can skip it in emergency.

Recent research in development and testing field has almost no interaction. But now an initiative have been taken by Europe researchers in Testnet Network[12] which takes both as a single unit. UML is also an example which was used only for modeling now also facilitating advance testing.

Design Specification for Testing

Ease to test a program and more effectiveness are features of testing which enhance its usefulness. A good test method tests the entire program e.g. Branch coverage.[13]

Phases of Testing

While testing a large system, testing is performed in steps. A subsystem is tested in each step. And then integration testing is performed while combining these subsystems in a consolidated system in incremental order. For a subsystem we can take different test cases depending upon the subsystem under consideration.

Recently specification and design based testing has been performed. Several authors gave emphasis on the use of architectural model[14,15] to derive testing. It not only derives test cases but also decomposes a system in subsystems and performs an Integration based testing. In the end, most practical testing i.e. Regression testing in which a set of already passed tests are again executed to find out that no new error have been introduced.

Testing a module based Development

In traditional engineering domain the latest paradigm for development is module based i.e. merging separately developed simple modules to built a complex large system. It is easy to test a module rather than a complex large system. Hence module based testing makes the evaluation easy.

While working with module based approach a new issue arises i.e. before testing understandability of what a modules really does? It is client capability to validate a module results.

Conclusion

It was general overview of facilitating the deployment of testing of modules. The problem of testing large system is eliminated by testing its subsystems at initial level. And considering development and testing as a single unit. Although testing is automated now but tester's expertise is also important for this challenging task. Some future research perspective is provided in [16].

References

[1] Bache,R., Mullerburg,M.: Measures of Testability as a Basis for Quality Assurance. Software Engineering Journal.5 (March 1990) 86-92.

[2] Bertolino,A., Corradini, F, Inverardi, P.,Muccini, H: Deriving Test Plans from Architectural Descriptions. Proc. Int. ACM Conf on Soft Eng(2001) 211-220.

[3] Bertolino,A., Inverardi, P.,Muccini H: An Explorative Journey from Architectural Tests Definition down to Code Tests Execution. Proc. Int. ACM Conf on Soft Eng(2001) 211-220.

[4] Bertolino,A.:Knowledge Area Description of Software Testing. Guide to the SWEBOK, Joint IEEE-ACM Software Engineering Coordinating Committee. (2001) online at :http://www.swebok.org

[5] Bertolino,A: Marre, M.: A General Path Generation Algorithm for Coverage Testing. Proc. 10th Int. Soft.Quality Week, San Francisco, CA (1997).

[6] Bernot, G.,Gaudel M.C.,Marre, B.:Software Testing based on Formal Specification: A Theory and A Tool. Software Eng Journal 6.(1991) 387-405.

[7] Dick,J.,Faivre, A.:Automating the Generation And Sequencing of Test cases from Model Based Specification(1993) 268-284.

[8] Harrold, M.J.: Testing: A Roadmap. In A. Finkelstein (Ed.), The Future of Software Engineering, ACM Press(2000) 63-72.

[9] Hierons, R., Derrick, J.:Special Issues on Specification Based Testing. Soft. Testing, Verification and Reliability 10(2000).

[10] Littlewood, B.,Popov, P.T.,Strignini, L.,Shryane, N.:Modelling the Effects of Combining Diverse Software Fault Detection Technique. IEEE Trans. Software Eng. 26(12)(2000) 1157-1167

[11] Myers, G.J.:The Art of Software Testing Wiley(1979).

[12] Lyu,M.R.(Rd.):Handbook of Software Reliability Engineering. IEEE Comp.Soc.Press/Mc Graw-Hill(1996).

[13] Latella, D.,Massink,M.: On Testing And Conformance Relations for UML statechart diagram behaviors.

[14] TESTNET-Integration Of Testing Methodologies. http://http://www.-lor.int-evry.fr/testnet/.

[15] Vegas, S.:Characterization schema for Software Testing Techniques.Int.Software Enginering Research Network(ISERN'01) strachclyde,Scotland(2001).

[16] Weyuker, E.J : On Testing Non Testable Programs. The Computer Journal 25(4)(1982) 465-470.