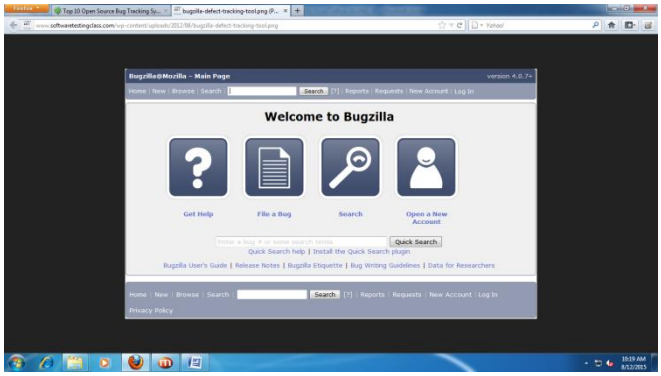


Figure 2: Snapshot of Bugzilla tool

Software Testing:

Ron Patton in his book Software Testing [4] defines testing as

The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.

A software quality assurance person's main responsibility is to create and enforce standards and methods to improve the development process and to prevent bugs from ever occurring. In real organizations, there the activities of testers**.**

Definition by IEEE [1]:

1. The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.
2. The process of analyzing a software item to detect the differences between existing and required conditions (that is bugs), and to evaluate the features of the software item.

Software testing is area that is a quite huge. It is not easy to describe it in a few sentences.

4.1 There are four software testing strategies:

Unit testing

It is done at the lowest level. It tests the basic unit of software, which can be a module or component. Unit is the smallest module i.e. smallest set of lines of code which can be tested. Unit testing is just one of the levels of testing which contribute to make the big picture of testing a whole system. Unit testing is generally considered as a white box test class.

Integration Testing

It is done when two or more tested units are combined into a larger structure. This testing is often done on the interfaces that are between the components and the larger structure that is being constructed, if its quality property cannot be properly assessed from its components.

System Testing

It tends to test the end-to-end quality of the entire system. System test is often based on the functional and requirement specifications of the system. Non-functional quality attributes, such as security, reliability, and maintainability, are also checked.

Acceptance Testing

It is done when the complete system is handed over to the customers or users from developer side.

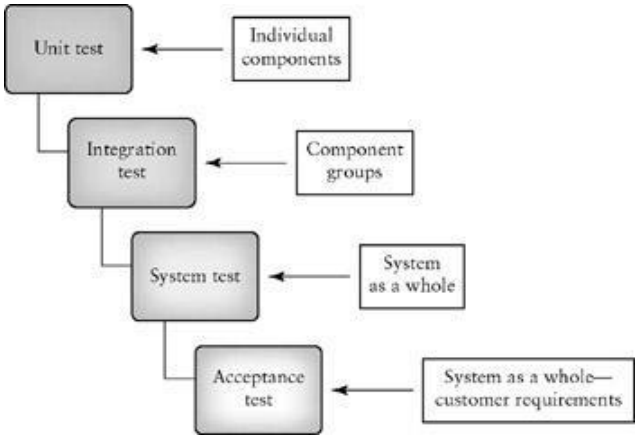
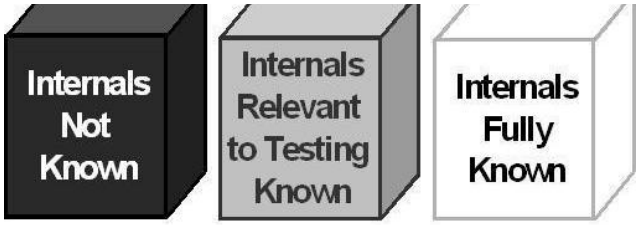


Figure 3: Software testing strategies

The aim of acceptance testing is to give assure that the system is working rather than to find errors.

4.2 Testing Methodology:



4.3 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:

1) Test a Program to Try to make it Fail: Testing is the process of executing a program with the intent of finding errors [5]. 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.

2) Start Testing Early: 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. Figure 1 depicts the increase in cost of fixing bugs detected/fixed in later phases.

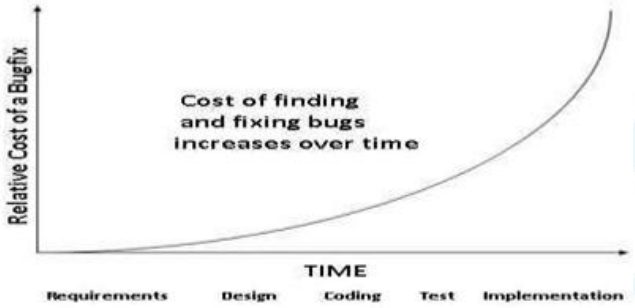


Figure 4: Depicts the increase in cost of fixing bugs detected/fixed in later phases

3) Testing is Context Dependant: Testing is done differently in different contexts. Testing should be appropriate and different for different points of time. For example, 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 [6]. 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.

4) Define Test Plan: Test Plan usually describes test scope, test objectives, tes 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.

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 [5]. These must be specified in a way that is measurable so that testing results are unambiguous.

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 [5]. Choose test inputs that possibly will uncover maximum faults by triggering failures.

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].

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 [8]. 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.

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 [5]. 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.

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 [9].

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 [7], so additional testing efforts

should be more for used on more error-prone sections until it is subjected to more rigorous testing.

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 [10]) 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)

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).

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 [12]. 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.

5. Conclusion

Quality is the main focus of any software engineering project. Without measuring, we cannot be sure of the level of quality in software. So the methods of measuring the quality are software testing techniques. In this paper i have discuss about how SELENIUM IDE tool helps to achieve quality and also describe how to communication take place between tester an developer? My thesis work mainly focuses on functional quality.

Reference

[1] IEEE Computer Society. IEEE Std. 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology (1990). Available from World, Wide Web: http://www.ieee.org/portal/site.
[2] ISTQB. International Software Testing Qualification Board ISTQB [online] (2009) [cited 2009-11-30]. Available from World Wide Web: http://www.istqb.org/index.htm
[3] Daniel Galin (2004). Software Quality Assurance, from theory to implementation. Pearson Education Limited.
[4] Ron Patton (2001). Software Testing. Sams Publishing
[5] Myers et al. ―The art of software testing‖, New York: Wiley, c1979. ISBN: 0471043281
[6] Shari Lawrence Pfleeger (2001). ―Software Engineering, Theory and Practice‖, Pearson Education
[7] James Bach (4/16/03). ―Exploratory Testing Explained‖, v.1.3
[8] Ian Somerville (2001). ‖ Software Engineering‖, Addison-Wesley
[9] Programming Research Ltd, ―Static and Dynamic Testing Compared‖.
[10] Antonia Bertolina (2003). ‖Software Testing Research and Practice‖, Proceedings of the abstract state machines 10th international conference on Advances in theory and practice, 1-21.
[11] Rajat Kumar Bal, ‖ Software Testing‖.
[12] Peter Sestoft (2008-02-25). ‖ Systematic software testing‖, Version 2