7.6 TESTING PROCESS

- Comparison of different Techniques
- Levels of Testing
- Test plan
- Test case specifications
- Test case execution and analysis
- Reliability models
- Source code metrics

Testing is the last phase in the life cycle of the software product, before the software product is developed. We even know that, software typically undergoes changes even after if has been delivered. In order to validate that a change has not affected some old functionality of the system, regression testing is done. In regression testing, old test cases are executed with the expectation that the same old results will be produced. But this testing adds on additional requirements on the testing phase.

Testing is the costliest activity in the software development. So, it has to be done efficiently. And testing cannot be done all a sudden, a careful planning is required and the plan has be executed properly. The testing process focuses on how testing proceed for a particular project.

Structural testing (White Box testing), is best suited for testing only one module (and not for entire program therefore it is difficult to generate test cases to get the desired coverage) It is good for detecting logic errors, computational errors and Interface errors. Functional Testing (Black - Box Testing) is best for testing the entire program or system. It is good for input errors and data handling errors.

Code Reviews

- Cost effective method of detecting errors
- Code is reviewed by a team of people in a formal manner. Here faults are found directly.
- Does not require test case planning, test case generation or test case execution.

7.7 TEST PLAN

It is a general document for the entire project that defines the scope, approach to be taken and the schedule of testing, test deliverables and the personnel responsible for different activities of testing. This plan can be prepared before actual testing begins and can be done in parallel with the coding and design phases.

The inputs for forming the test plan are :

(1) Project plan

(2) Requirements document

(3) System design document

A test plan containing the following:

- Test unit specification.
- Features to be tested
- Approach for testing
- Test deliverables
- Schedule-Personnel allocation.

Test unit specification:

- An important activity of the test plan is to identify the test units. A test unit is a set of one or more modules together with data.
- Along with identification of test units, different levels of testing are also specified like unit testing, integration testing etc.
- While forming "test units", it is input to look for "testability" of a unit. (*i.e.*, a unit should betested early or it should be possible to form meaningful test cases and execute the unit without much effort with these test cases).

7.8 DEBUGGING

A practicing programmer inevitably spends a lot of time tracking down and fixing bugs. Debugging, particularly debugging of other people's code, is a skill separate from the ability to write programs in the first place. Unfortunately, while debugging is often practiced, it is rarely taught. A typical course in debugging techniques consists merely of reading the manual for a debugger.

Identify the Bug

Debugging means removing bugs from programs. A bug is unexpected and undesirable behaviour by a program.

Occasionally there is a formal specification that the program is required to follow, in which case a bug is a failure to follow the spec. More frequently the program specification is informal, in which case people may disagree as to whether a particular program behavior is in fact a bug or not.

As one of the people writing a program, you should be a source of bug reports. Don't simply rely on testers or users. If you notice something odd while running a program, it can be tempting to disregard it and hope that it will go away. This is especially true if you are working on something else at the time. Resist the temptation. Record the bug. If you have data files or logs, save them. If you have time later, return to the issue; otherwise, pass it on to somebody else. As one of the people most familiar with how the program is supposed to work, you are in the best possible position for detecting unexpected behaviour.

Once you have a bug report, the first step in removing the bug is identifying it. This is of particular importance when working with a bug report produced by somebody else, such as the testing team or a user. Some bugs are relatively obvious, as when the program crashes unexpectedly. Others are obscure, as when the program generates output which is slightly incorrect.

Many bug reports received from users are of the form "I did such and such and something went wrong." Before doing anything else, you must find out what went wrong—that is, you must identify the bug by determining the program behaviour which was unexpected and undesirable. Any attempt to fix the bug before understanding what went wrong is generally wasted time.

Identifying a bug reported by a user typically requires getting the answer to two questions: "What did the program do?" and "What did you expect the program to do?" The goal is to determine precisely the behaviour of the program which was unexpected and undesirable.

Once the bug has been identified, the easiest and fastest way to fix it is to determine that it is not a bug at all. If there is a formal specification for the program, you may have to modify the specification. In other cases, you may have to modify the expectations of the user. This rapid fix is often known as declaring the behaviour to be an "undocumented feature." Despite the obvious potential for abuse, this is in fact sometimes the correct way to handle the problem.

Unfortunately, most bugs are real bugs, and require further work.

Replicate the Bug

The first step in fixing a bug is to replicate it. This means to recreate the undesirable behaviour under controlled conditions. The goal is to find a precisely specified set of steps which demonstrate the bug.

In many cases this is straightforward. You run the program on a particular input, or you press a particular button on a particular dialog, and the bug occurs. In other cases, replication can be very difficult. It may require a lengthy series of steps, or, in an interactive program such as a game, it may require precise timing. In the worst cases, replication may be nearly impossible.

Programmers are sometimes tempted to skip the replication step, and to go straight from the bug report to the fix. However, failing to replicate the bug means that it is impossible to verify the fix. The fix may turn out to be for a different bug, or it may have no significant effect at all. If the bug has not been replicated, there is no way to tell.