SE Practices: Simple Steps to Achieve SQA

Subhash. G. Desai

Understanding of Quality Concepts and Software Testing practices is vital and ingredient part of any Software product. There are no shortcuts to achieve high degree of quality, at the same time, to meet the deadlines and completing projects on time, it is necessary to define methodology and adequate guidelines to follow.

We shall attempt to cover major areas leading to our success and formulate strategy to overcome above stated issues.

1. Quality Concepts:

Quality is not Expensive, It is priceless.

- Mohandas K Gandhi

Very old concept and very basic to every human who can identify Good or Bad Quality. Quality is applicable to every thing a person can see

- Person
- Product
- Events

We shall make following assumptions

- We are people
- Learning is an interactive and iterative process
- Small changes can produce BIG results
- Silence means non-involvement
- Today's problems come from Yesterday's solutions
- Dividing an Elephant in half does not produce 2 Elephants
- The number of Programmers in IT department is directly proportional to the errors it produces
- When you conduct a SQA training program, the person who needs it most will probably be too busy to attend it
- The most defective software in a lot will attracts maximum attention

1.1 QUALITY: THE THREE PILLARS

Prevention

Tries to assure that everything is right (optimal) the first time, or that the quality of an existing system doesn't deteriorate.

Defect Removal

Tries to find as many defects as possible as early as possible.

Improvement and Feedback
Tries to change the way of working, resulting quality should be better without use of more resources. Learn from errors.

1.2 ERRORS, DEFECTS AND FAILURES

An error is an oversight or a wrong decision made in development. Errors lead to defects (or faults) in software. Defects lead to failures during run time.

Examples of FAILURES:
The product doesn’t comply with documentation.
The product doesn’t meet reasonable customer expectations.
The product is too slow or lacking other important properties.
The product just doesn’t work.

Specifications	Anti-Specifications
What the product Must do	What the product must not do

Define which failures should not occur.

2. Testing:

We presume that there are defects. Testing is not only the execution of programs. It means all actions performed to remove defects. The earlier you detect defects, the better. Projects get delayed during the test phase.

Errors should not be detected by user. Defects discovered by user can ruin our reputation. Humans are blind to their own errors. Testing is a destructive process. Testing should focus on special cases. You can prepare a test as soon as you know what you have to do. Test must be prepared by a second person.

2.1 Testing work steps:

- Analysis of Specification, Determine test Candidates, Detail Plan
- Test Design
- Test Implementation, Test Execution
- Check for termination
- After work

Project Leader should handle analysis of specification and determine test Candidates using listed Input and generate stated Results.

Input : Requirement Specification, Possibly design documents, Test candidate list

Results : List of test candidates, List of clarification requests, Specific test planning

Tester should handle Test Design using listed Input and generate stated Results.

Input : Requirement Specification, Possibly design information, Everything produced so far.
Results : Test plan, possibly with more test candidates, Test cases, test tool and program descriptions

Tester or Programmer should handle Test Implementation, Test Execution using listed Input and generate stated Results.

Input : Everything produced before
Results : Verified test data structure, Test support resources, Test items, Test log, Possibly revised test specifications, Possibly revised test data

Project Leader or Tester should handle Check for termination using listed Input and generate stated Results.

Input : Test log, Test Specification (completeness and termination requirements)
Results : Information in the test summary, Possibly revised test specifications, data and log .

Programmer and Tester should handle After work and generate stated Results.

Results : Complete test summary report, Test statistics, Stored record of test results.

2.2 Black Box Vs White Box Test:

Functional test (Black Box)
The test planner knows the specification only. Tests are prepared to cover the specification.
Structure test (White Box)
The test is designed to cover all statements, branches/paths in the code. No method whatsoever can guarantee that all errors are found. But we may find 60% per test step.

2.3 STOP criteria:

When to stop testing?

- Stop when the scheduled time for testing runs out.
- Stop when all the test cases execute without error.
- These criteria are essentially useless.

Reasonable Stop criteria:

- Require black box testing
- Require a certain coverage
- You may seed errors and stop when a certain percentage of them are detected.
- Record the errors found and stop when testing becomes unproductive.
- Be critical and use judgment, based on earlier test experience.
- Systematic learning from all errors:
- Record all defects, analyze causes.
- Do something about the defect causes.
- Include field errors and suggestions into the process.
- Defect and failure data are the most important source for improvement.

2.4 Test Preparation guidelines:

Module Test

- Ask the designers if you have questions. Don't assume anything.
- Many inputs are hidden: not mentioned explicitly.
- Is the screen as shown in the user manual? Follows standard?
- Standard for dialogues followed? Defaults right?
- Possibility to quit and redo.
- Erroneous input does not break the system. Error handling and messages.
- Perform boundary value analysis.
- Error guessing. Assume an error --- > design test cases to find it.
  Error guessing needs 'destructive' thinking (Empty file).
  - The test must be repeatable.
- Any priority for what to test?
  No priority, but full coverage.
- To test <u>changed</u> code, rerun all old test cases.

Integration Test

- Decide on the order of Program integration.
  Top-down : Main program first. Exit immediately. Log that module is entered.
  Provide constant input. Provide random input. Read input values from terminal or file. Provide debugging facilities.
  Bottom-up : Lowest level first. Drivers. Call module under test.
  Provide required data. Log outputs.
- In both cases, start with complicated module.
- Combination: from both sides.
- Modules as they are ready: No planning at all.
- Decide on how many modules to integrate for every step.
- Execute every call at least once.
- Check that all global data are defined and that the definition is unambiguous.
- Every job step to be executed at least once.

Choose test data to detect following errors:

- Functional misunderstanding.
- Sequencing of functions wrong.
- Violation of boundaries.
- Data type inconsistency – every program must use the same data type.
- Speed problems – too high or low speed at transmissions.
- Format inconsistencies.
- Violation of conventions and restrictions.
- Wrong interpretation of data.
- Number of parameters – to be the same number, or between boundaries if variable. Order of parameters.
- Cleaning up after execution – Are resources released, scratch files deleted etc?
- Data flow anomalies – Missing data, disappearing data, not used data, wrong use of defaults.
- Integration test must be done stepwise.

System Test

During this phase following errors are to be detected:

- Discrepancies between system and specification.
- Misunderstood requirements.
- Usability problems.
- Performance problems.
- Installation problems.
- During maintenance, side effects of changes are one of the most difficult problems.
- The best remedy is to rerun existing test cases. This is called regression test.
- Tests are derived from usage situations, not from the system structure.
- Concentrate on normal data, but complicated situations.
- If this is a new release, or a new system planned to replace an existing one, concentrate on the capabilities the OLD system had.
- Users will not accept that old functions cease to work.
- Volume testing: Maximum amounts of data, More than Maximum, Random or old production data, fill all buffers and files.

3.0 How to handle time pressure:

- System test is your last chance.
- Early test preparation most important.
- Reduction in test – loss in quality.
- Make consequences of test reduction clear.
- Consequences for maintenance?

  Regression test will be more costly or impossible.

- Priority
  Which regression test cases to run?
  The safest is to run all. Costs too much.
  Run a total regression test at certain time intervals.
- Have confidence in your system.
- Get to know the risk areas.
- Test must reflect real life situations.
- Continue updating and expanding the test system.
- Finally Regression test will still be patchwork. But, after all, this has been reality for all the time.

References:

1. Establishing Quality Assurance Function in System Development
   Baylor University
2. Software Quality Assurance of Management Processes using Core Measures
   QSM – Quantitative Software Management
   September 2001
3. Project Management Best Practices
   RL Information Consulting LLC
4. Tips for using Microsoft Project and Project Server
   Tech Republic Inc.
   February 2003
5. Do the Right Projects, The Right way
   Artemis Management Systems
   November 2001
6. Rules for Project Success: Don’t Ignore your Sense 111
   Scitor Corporation
7. Management by Projects
   Alert
   August 2000
8. Knowledge-Based Project Planning
   Naval Research Laboratory
   June 2001