

Introduction to Software Quality Assurance & Testing

Sources:

Code Complete, 2nd Ed., Steve McConnell

Software Engineering, 5th Ed., Roger Pressman

Testing Computer Software, 2nd Ed., Cem Kaner, et. Al.

Software Quality Assurance

- The purpose of SQA is to find and report defects AND succeed in getting them fixed
- What is a software defect?
 - Definition #1: There is a mismatch between the program and its requirements spec or functional spec
 - This definition is fine if a requirements specification exists and is complete and correct (not always true)
 - Definition #2: The program does not do what its end user reasonably expects it to do
 - This definition always applies, even when there's no specification

Software Quality Assurance

- Categories of Defects
 - Functional defects
 - The program's features don't work as they should
 - User Interface defects
 - Usability problems
 - Performance defects
 - Too slow, Uses too much memory/disk space/bandwidth/etc.
 - Error Handling defects
 - Failure to anticipate and handle possible errors, or deal with them in a reasonable way
 - Security defects
 - Attackers can compromise the system and access sensitive data or other resources

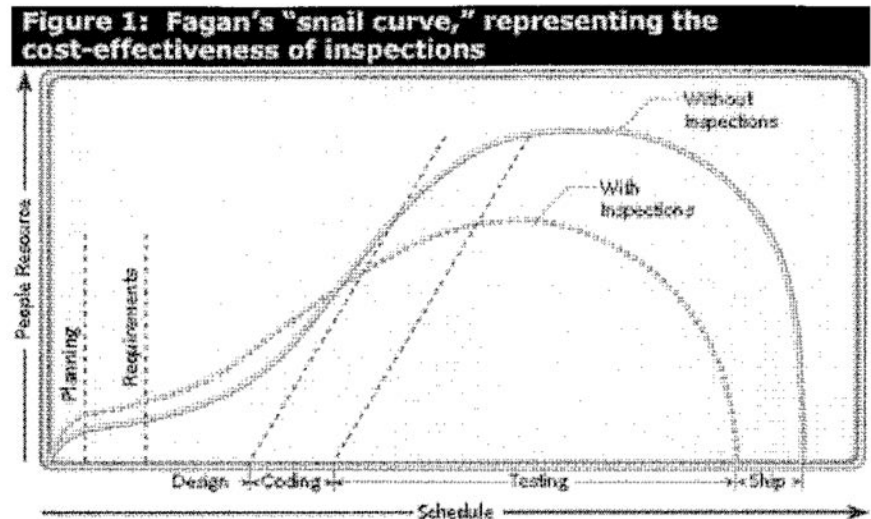
Software Quality Assurance

- Categories of Defects
 - Load defects
 - Can't handle many concurrent users, can't handle large data sets
 - Configuration defects
 - Doesn't work on the required hardware/OS/browser configurations
 - Race conditions
 - Behavior depends on the interleaving of concurrent activities
 - Documentation defects
 - User manuals or online help isn't clear, complete, well-organized

Software Quality Assurance

- The longer defects remain in the system, the more expensive they become
 - The cost of a defect grows dramatically the longer it remains in the system
 - What is the cost of a defect in the requirements specification if it's found found

- during requirements phase?
- during implementation?
- after product ships?



- SQA should be performed throughout the software development life cycle
 - It's not something you do only at the end after everything's pretty much done

Verification & Validation

- Validation
 - Build the right product
 - Those activities that attempt to determine that customer needs can be met by a product
- Verification
 - Build the product right
 - Transformational activities that are performed at each step of the product life cycle
 - Evaluating at each stage of the life cycle to ensure the software meets the requirements
- Focus
 - Completeness, Consistency, Feasibility, Testability

Software Quality Assurance

- The two primary SQA activities:
 - Technical Reviews
 - Software Testing

Technical Reviews

- A “review” is a meeting where a work product is reviewed by a small group of people who are qualified to give feedback, find problems, suggest improvements, etc.
- Anything can be reviewed: requirements spec, functional spec, design, code, test plan, user documentation
- Reviews range in formality
 - In the morning, spend some time reviewing your work of the previous day
 - Informal requests for feedback from peers
 - Pair programming
 - Formal meetings, pre-scheduled, specific invitees, prior preparation
- Problems found during reviews are fixed, resulting in improved quality
- Reviews are the most effective QA technique, both in terms of cost and number of defects discovered

Review Meetings

- Review the product – not the producer
- Set an agenda and keep it
- Limit debate and rebuttal
- Enunciate problem areas but don't try to fix anything
- Take written notes
- Limit the number of participants
- Insist upon advance preparation
- Develop a checklist
- Allocate resources and time schedules
- Conduct meaningful training

Software Testing

- Testing is the process of detecting errors by running the actual software and verifying that it works as it should
 - Test cases, Expected results, Actual results
- Testing is by far the most popular QA activity (but not the most effective)
- Formal technical reviews are cheaper and more effective than testing, but are often ignored
- Research has shown that all forms of testing combined usually find less than 60% of the errors present
- A typical project might expend 50% of its resources on testing

Software Testing

- Exhaustively testing software is not feasible
 - The number of possible input combinations is effectively infinite
 - The number of unique paths through the code is effectively infinite
 - You might not live long enough to exhaustively test a non-trivial software system
- We must do partial testing because we only have enough resources (time and money) to run relatively few test cases
- Partial testing can never prove the absence of defects
 - If the system passes all your test cases, there could still be defects, you just need more or better test cases to find them

Software Testing

- Effective testing lies in intelligently choosing the relatively few test cases that will actually be executed
 - Test all requirements and features defined in the requirements spec. and functional spec.
 - Focus on scenarios that users are likely to encounter in practice
 - Test cases should not be redundant (i.e., each one should follow a different path through the code)
 - Analyze the program's design and code to find potential weak areas
 - Analyze all points at which data enters the system and look for ways to attack it
 - Code coverage

Software Testing

- Approaches for test case design are generally divided into two broad categories: Black Box Testing and White Box Testing
- Black Box Testing
 - The tester has limited knowledge of the inner workings of the item being tested
 - Test cases are based on the specification of the item's external behavior
- White Box Testing
 - The tester has knowledge of the inner workings of the item being tested
 - Test cases are based on the specification of the item's external behavior AND knowledge of its internal implementation

Software Testing

- Testing is unlike other software development activities because the goal is to break the software rather than to create it
- Effective testing requires the assumption that you will find defects
- Effective testing requires that you want to find defects
- If you think you won't find defects, or you don't want to, you will have set up a self-fulfilling prophecy
- Testing by both developers and an independent testing group are essential
 - They have different perspectives and motivations
 - They do different kinds of tests (developer does white box, test team does black box), which tend to discover different types of defects

Software Testing

- Defects are not evenly distributed (i.e., they tend to cluster)
- Research has shown that:
 - 80% of a system's defects are found in 20% of its code
 - 50% of a system's defects are found in 5% of its code
- There is a high correlation between bugs and complex code.
 - Use tools to measure code complexity, and focus testing on those modules with the most complex code
- One goal of testing is to identify the most problematic modules
 - Redesign may be needed if there is an inherent design flaw
 - Or, replace buggy module with a third-party library/product

Software Testing

- How many defects should you expect to find?
 - It depends on your development process
 - Most projects experience between 1 and 25 errors per 1000 LOC
 - The Applications Division at Microsoft reports 10 to 20 errors per 1000 LOC, with 0.5 errors per 1000 LOC in released products

Software Testing

- Automation of test cases is essential to make frequent re-running of test cases feasible
- A lot of the interesting testing work is found in inventing and creating ways to automate test cases (i.e., create programs whose purpose is to test other programs)
- Automation requires a lot of software design and implementation (sometimes called “Test Engineering”)
- Some tests are difficult to automate and must be run manually

Unit Testing

- Exercise a specific module in a controlled environment
- Typically involves
 - Scaffolding
 - Stubs and drivers
- Stubs
 - Modules below a unit to mimic behavior of dependent modules
- Drivers
 - Modules above a unit that drive the unit in the same fashion its calling modules do
- Involves both white box and black box testing

Integration Testing

- After unit testing – put them together to do testing
- Top down or bottom up
- Tester needs to understand the behavior of integrated modules
 - Module hierarchy can help

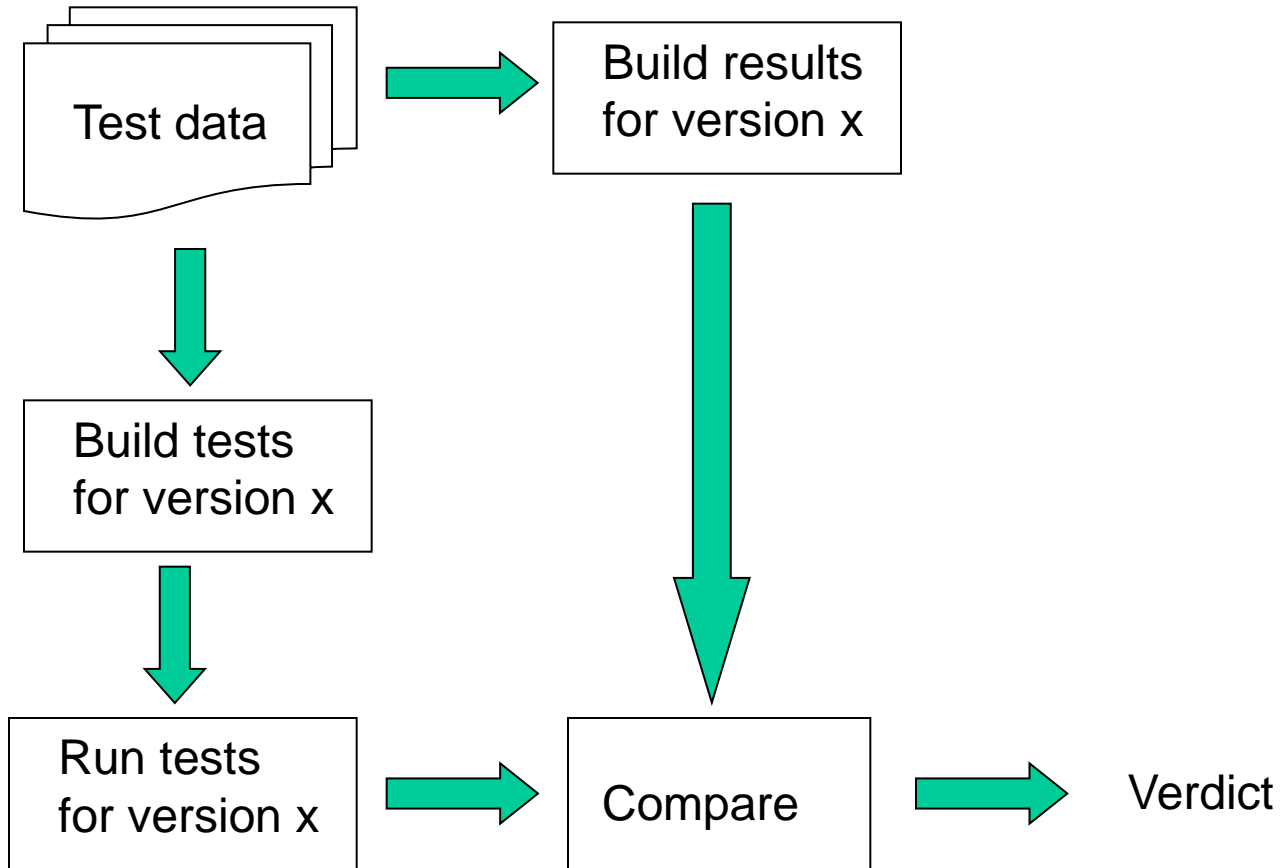
System Testing

- Execution of the entire system
 - Does it conform to the overall system requirements
 - Depends on the document
- May be simulated
 - Depends on accuracy of the simulator
- Tests
 - Functionality
 - Performance
 - Reliability
 - Usability

Regression Testing

- Check to see that an update does not re-introduce errors
- Check
 - Functionality – typically black box tests
 - Architecture – gray box tests (some knowledge of inner workings)
- Typically a large suite of tests
 - All functionality
 - All changes
- Needs to be automated

Regression Testing



Automating Regression Testing

- Challenging
- What parts of program output should be checked?
- Simple but annoying issues
 - Use of dates in output
 - Changes in whitespace
 - Format changes
 - Lead text changes
- Answer
 - Don't use complete output
 - Just extract the relevant information

Regression Testing

- Keep it updated!
 - Bug fixes – tests to ensure they stay fixed
 - Functionality additions
 - Platform changes
 - Etc.
- If you branch the code, you must branch the regression tests

Formal Verification

- In addition to Technical Reviews and Software Testing, Formal Verification is another approach to QA
- Create a formal “model” of the system
 - Some kind of automaton (i.e., state machine) or other mathematical abstraction that precisely captures the system’s behavior
- “Check” the model by formally proving that it implements the desired behavior
 - Automated theorem proving systems are often applied
 - Or, prove that the model does not behave correctly, thus revealing a defect
- Historically, formal verification has been expensive and limited to relatively small programs, but techniques are improving all the time. Challenges include:
 - Complex systems are hard to formalize
 - State space explosion: real systems have so many possible states that proving things about them is hard
 - Ensuring that the “model” accurately captures the system’s behavior
 - Making it accessible to people who aren’t formal verification experts