Software Quality Assurance, Part 1

Charles Carter

October 31, 2017

Table of Contents

Software Testing Strategies

Generic characteristics

- Conduct formal technical reviews (requirements, design, code, and testing)
- Work upward from “lower” components to “higher” components
- Use different testing techniques at different stages, as appropriate
- Who tests: code authors and independent testers
- Finding and correcting defects are different activities

Strategy

- Test low level components (individual functions, methods, etc.)
- Test intermediate level components (modules, classes, libraries, etc.)
- Test high level components (entire programs, applications, systems)

Strategy

- Verification — “Did we build the software right?”
- Validationn — “Did we build the right software?”

Strategy

- Developers — have a vested interest in proving that the software is correct
- Testers — have a vested interest in proving that the software is *NOT* correct

Conventional testing

- Spiral view — work from inner (lower) components to outer (higher) components: unit testing, integration testing, validation testing, system testing
- Block view — work with building blocks: individual functions and methods, classes and modules, packages and modules, applications and systems

Object oriented testing

- Discovery of errors in analysis and design
- Errors in models and patterns used
- Errors in messaging and communication between objects
- Test communication and collaboration, side effects of new classes

Strategic Issues

Strategic Issues

- Specify products in a quantifiable manner before testing
- State testing objectives explicitly
- Develop and use profiles for each user category
- Emphasize *rapid cycle testing*

Strategic Issues

- Build software designed to test itself
- Conduct technicaljreviews *prior* to testing
- Conduct technical reviews of testing processes as well as code
- Develop process improvement plans for testing

Unit Tests

Unit tests

- Responsibility of developer
- Design test before writing code
- Testing of smallest discrete units of software
- Testing within boundaries of individual components
- Test all basis paths
- Test all data structures

Unit tests

- Interface — does data properly enter and exit component
- Local data structures — do they fit the data
- Boundary conditions — many errors occur at boundaries
- Independent paths — test all paths (within reason)
- Error handling paths — test exceptions code

Integration Tests

Integration Tests

Integration testing . . .

- . . . is a systematic technique for constructing the software architecture while conducting tests to uncover interfacing issues
- . . . is done incrementally (in small batches) where errors are easier find and correct

Top down integration

Start with the top level component, e.g., `public static Main(string args[])`, using stubs to simulate functionality of uncompleted lower components. Mirrors top-down development, stepwise refinement. Replace stubs with completed components.

Bottom up integration

Start with the atomic units, using drivers to simulate unwritten higher level components. Combine tested components into composite components, implementing drivers. Mirrors functional programming techniques, where software is viewed as a series of transformations.

Regression testing

New components break old components. Answer the question: “Does what I am writing break anything that I have written?” As software develops, regression testing becomes more elaborate and takes more effort. Write representative tests, focus on changes, focus on error types.

Smoke (sanity) testing

Continuous integration calls for continuous testing, daily or even more often. Create a build server, and build applications at least daily. The object is to catch “show stopper” errors.

Validation Testing

Validation testing

After unit testing, integration testing, and regression testing, the software is compiled and packaged as a single system. At this point, the question is: “Did we build the right software?” That is, does the product conform to the requirements as specified by the customer?

Validation test criteria

Validation tests are targeted to the requirement specifications. Develop criteria for validating the software. Establish metrics that demonstrate the correctness (or incorrectness) of the software. Are the functional requirements satisfied? Are the behavioral requirements satisfied? Are the performance requirements satisfied? If not, measure the deficiences. Extablish exactly the nature of the failures.

Configuration review

Configuration items consist of initialization variables, authentication credentials, routing parameters, initial settings and the like. Testers should audit these to ensure correctness.

Alpha and beta testing

Alpha testing of software occurs with the collaboration of developers and ordinary users. Beta testing occurs in the field, outside the presence of the developers. Alpha and beta tests are used to see how ordinary useers will actually use the software.

System Testing

System testing

- Recovery testing
- Security testing
- Stress testing
- Performance testing

Recovery testing

Software will fail! How long does it take for the software to recover? How long does it take to fix the problems that caused the failure? Recovery testing forces the software to fail in an effort to answer these questions.

Security testing

Criminals will subvert software? What features of the software permit attacks? What data is subject to risk? What functionality will be impaired? Security testing attempts to answer these questions.

Stress testing

Abnormal conditions occur? How does the software perform in adverse conditions? At what point does the software cease to function? How does the software handle hardware failures? . . . connectivity issues? . . . network latency? Stress testing attempts to answer these questions.

Performance testing

Does the software meet performance criteria? What is the time complexity of the various components? What is the space complexity of the various components? Where are the bottlenecks?