Software Testing

Observations about Testing

- “Testing is the process of executing a program with the intention of finding errors.” – Myers
- “Testing can show the presence of bugs but never their absence.” - Dijkstra

Good Testing Practices

- A good test case is one that has a high probability of detecting an undiscovered defect, not one that shows that the program works correctly
- It is impossible to test your own program
- A necessary part of every test case is a description of the expected result

Good Testing Practices (cont’d)

- Avoid nonreproducible or on-the-fly testing
- Write test cases for valid as well as invalid input conditions.
- Thoroughly inspect the results of each test
- As the number of detected defects in a piece of software increases, the probability of the existence of more undetected defects also increases

Good Testing Practices (cont'd)

- Assign your best people to testing
- Ensure that testability is a key objective in your software design
- Never alter the program to make testing easier
- Testing, like almost every other activity, must start with objectives

Levels of Testing

- Unit Testing
- Integration Testing
- Validation Testing
  - Regression Testing
  - Alpha Testing
  - Beta Testing
- Acceptance Testing

Unit Testing

- Algorithms and logic
- Data structures (global and local)
- Interfaces
- Independent paths
- Boundary conditions
- Error handling

Why Integration Testing Is Necessary

- One module can have an adverse effect on another
- Subfunctions, when combined, may not produce the desired major function
- Individually acceptable imprecision in calculations may be magnified to unacceptable levels

Why Integration Testing Is Necessary (cont'd)

- Interfacing errors not detected in unit testing may appear
- Timing problems (in real-time systems) are not detectable by unit testing
- Resource contention problems are not detectable by unit testing

Top-Down Integration

1. The main control module is used as a driver, and stubs are substituted for all modules directly subordinate to the main module.
2. Depending on the integration approach selected (depth or breadth first), subordinate stubs are replaced by modules one at a time.

Top-Down Integration (cont’d)

3. Tests are run as each individual module is integrated.
4. On the successful completion of a set of tests, another stub is replaced with a real module
5. Regression testing is performed to ensure that errors have not developed as result of integrating new modules

Problems with Top-Down Integration

- Many times, calculations are performed in the modules at the bottom of the hierarchy
- Stubs typically do not pass data up to the higher modules
- Delaying testing until lower-level modules are ready usually results in integrating many modules at the same time rather than one at a time
- Developing stubs that can pass data up is almost as much work as developing the actual module

Bottom-Up Integration

- Integration begins with the lowest-level modules, which are combined into clusters, or builds, that perform a specific software subfunction
- Drivers (control programs developed as stubs) are written to coordinate test case input and output
- The cluster is tested
- Drivers are removed and clusters are combined moving upward in the program structure

Problems with Bottom-Up Integration

- The whole program does not exist until the last module is integrated
- Timing and resource contention problems are not found until late in the process

Validation Testing

- Determine if the software meets all of the requirements defined in the SRS
- Having written requirements is essential
- Regression testing is performed to determine if the software still meets all of its requirements in light of changes and modifications to the software
- Regression testing involves selectively repeating existing validation tests, not developing new tests

Alpha and Beta Testing

- It’s best to provide customers with an outline of the things that you would like them to focus on and specific test scenarios for them to execute.
- Provide with customers who are actively involved with a commitment to fix defects that they discover.

Acceptance Testing

- Similar to validation testing except that customers are present or directly involved.
- Usually the tests are developed by the customer

Test Methods

- White box or glass box testing
- Black box testing
- Top-down and bottom-up for performing incremental integration
- ALAC (Act-like-a-customer)

Test Types

- Functional tests
- Algorithmic tests
- Positive tests
- Negative tests
- Usability tests
- Boundary tests
- Startup/shutdown tests
- Platform tests
- Load/stress tests

Concurrent Development/ Validation Testing Model

- Conduct informal validation while development is still going on
- Provides an opportunity for validation tests to be developed and debugged early in the software development process
- Provides early feedback to software engineers
- Results in formal validation being less eventful, since most of the problems have already been found and fixed

Validation Readiness Review

- During informal validation developers can make any changes needed in order to comply with the SRS.
- During informal validation QA runs tests and makes changes as necessary in order for tests to comply with the SRS.

Validation Readiness Review (cont’d)

- During formal validation the only changes that can be made are bug fixes in response to bugs reported during formal validation testing. No new features can be added at this time.
- During formal validation the same set of tests run during informal validation is run again. No new tests are added.

Entrance Criteria for Formal Validation Testing

- Software development is completed (a precise definition of "completed" is required.
- The test plan has been reviewed, approved and is under document control.
- A requirements inspection has been performed on the SRS.
- Design inspections have been performed on the SDDs (Software Design Descriptions).

Entrance Criteria for Formal Validation Testing (cont'd)

- Code inspections have been performed on all "critical modules".
- All test scripts are completed and the software validation test procedure document has been reviewed, approved, and placed under document control.
- Selected test scripts have been reviewed, approved and placed under document control.

Entrance Criteria for Formal Validation Testing (cont'd)

- All test scripts have been executed at least once.
- CM tools are in place and all source code is under configuration control.
- Software problem reporting procedures are in place.
- Validation testing completion criteria have been developed, reviewed, and approved.

Formal Validation

- The same tests that were run during informal validation are executed again and the results recorded.
- Software Problem Reports (SPRs) are submitted for each test that fails.
- SPR tracking is performed and includes the status of all SPRs (i.e., open, fixed, verified, deferred, not a bug)

Formal Validation (cont'd)

- For each bug fixed, the SPR identifies the modules that were changed to fix the bug.
- Baseline change assessment is used to ensure only modules that should have changed have changed and no new features have slipped in.
- Informal code reviews are selectively conducted on changed modules to ensure that new bugs are not being introduced.

Formal Validation (cont'd)

- Time required to find and fix bugs (find-fix cycle time) is tracked.
- Regression testing is performed using the following guidelines:
  - Use complexity measures to help determine which modules may need additional testing
  - Use judgment to decide which tests to be rerun
  - Base decision on knowledge of software design and past history

Formal Validation (cont'd)

- Track test status (i.e., passed, failed, or not run).
- Record cumulative test time (cumulative hours of actual testing) for software reliability growth tracking.

Exit Criteria for Validation Testing

- All test scripts have been executed.
- All SPRs have been satisfactorily resolved. (Resolution could include bugs being fixed, deferred to a later release, determined not to be bugs, etc.) All parties must agree to the resolution. This criterion could be further defined to state that all high-priority bugs must be fixed while lower-priority bugs can be handled on a case-by-case basis.

Exit Criteria for Validation Testing (cont’d)

- All changes made as a result of SPRs have been tested.
- All documentation associated with the software (such as SRS, SDD, test documents) have been updated to reflect changes made during validation testing.
- The test report has been reviewed and approved.

Test Planning

- The Test Plan – defines the scope of the work to be performed
- The Test Procedure – a container document that holds all of the individual tests (test scripts) that are to be executed
- The Test Report – documents what occurred when the test scripts were run

Test Plan

- Questions to be answered:
  - How many tests are needed?
  - How long will it take to develop those tests?
  - How long will it take to execute those tests?
- Topics to be addressed:
  - Test estimation
  - Test development and informal validation
  - Validation readiness review and formal validation
  - Test completion criteria

Test Estimation

- Number of test cases required is based on:
  - Testing all functions and features in the SRS
  - Including an appropriate number of ALAC (Act Like A Customer) tests including:
    - Do it wrong
    - Use wrong or illegal combination of inputs
    - Don’t do enough
    - Do nothing
    - Do too much
  - Achieving some test coverage goal
  - Achieving a software reliability goal

Considerations in Test Estimation

- Test Complexity – It is better to have many small tests that a few large ones.
- Different Platforms – Does testing need to be modified for different platforms, operating systems, etc.
- Automated or Manual Tests – Will automated tests be developed? Automated tests take more time to create but do not require human intervention to run.

Estimating Tests Required

SRS Reference	**Estimated Number of Tests Required**	**Notes**
4.1.1	3	2 positive and 1 negative test
4.1.2	2	2 automated tests
4.1.3	4	4 manual tests
4.1.4	5	1 boundary condition, 2 error conditions, 2 usability tests
…		
Total	165	

Estimated Test Development Time

Estimated Number of Tests: 165

Average Test Development Time: 3.5 (person-hours/test)

Estimated Test Development Time: 577.5 (person-hours)

Estimated Test Execution Time

Estimated Number of Tests: 165

Average Test Execution Time: 1.5 (person-hours/test)

Estimated Test Execution Time: 247.5 (person-hours)

Estimated Regression Testing (50%): 123.75 (person-hours)

Total Estimated Test Execution Time: 371.25 (person-hours)

Test Procedure

- Collection of test scripts
- An integral part of each test script is the expected results
- The Test Procedure document should contain an unexecuted, clean copy of every test so that the tests may be more easily reused

Test Report

- Completed copy of each test script with evidence that it was executed (i.e., dated with the signature of the person who ran the test)
- Copy of each SPR showing resolution
- List of open or unresolved SPRs
- Identification of SPRs found in each baseline along with total number of SPRs in each baseline
- Regression tests executed for each software baseline

Validation Test Plan
IEEE – Standard 1012-1998

1. Overview
    a. Organization
    b. Tasks and Schedules
    c. Responsibilities
    d. Tools, Techniques, Methods
2. Processes
    a. Management
    b. Acquisition
    c. Supply
    d. Development
    e. Operation
    f. Maintenance

Validation Test Plan
IEEE – Standard 1012-1998 (cont'd)

3. Reporting Requirements
4. Administrative Requirements
5. Documentation Requirements
6. Resource Requirements
7. Completion Criteria