Formal Software Testing

Terri Grenda, CSTE
IV&V Testing Solutions, LLC
www.ivvts.com

Scope of Testing

- Find defects early
- Remove defects prior to production
- Identify Risks
- Unbiased opinion

When Should Testing Occur

In all phases of the Software Development Life Cycles

- Requirements
  - Determine Verification Approach
  - Determine Adequacy of Requirements
  - Generate Test Plan Outline

When Should Testing Occur

- Design
  - Determine consistency of design with requirements
  - Determine adequacy of design
  - Generate structural and functional outline Test Plan
- Build (Code)
  - Determine consistency of design

When Should Testing Occur

- Test
  - System Test the application
  - Careful control and management of test environment
- Installation
  - Test installation process
  - Ensure correct versions of the program
  - Lessons learned passed onto supporting parties

When Should Testing Occur

- Neither development nor verification is a straight-line process
- Analyze the structures produced at this phase for internal testability and adequacy
- Determine that the structures are consistent with structures produced during previous phases
- Refine or redefine test scripts generated in previous phases

Development of Test Plan

- Identify the system development process
- Select and rank test objectives
- Identify the business risks associated with the system under development
- Place risks in matrix
- Determine test approach

Common Problems Associated with Testing

- Failing to define testing objectives
- Using ineffective test techniques
- Incomplete Statement of Work “The Goal”
- Changes in Technology
- Limited Tester Skills

Technological Development

- Integration
- System Chains
- Domino Effect
- Reliance on Electronic Evidence
- Multiple Users

Common Software Risks

- Unauthorized transactions can be accepted by the system
- Security of the system will be compromised
- Processing will not comply with organizational policy or government regulations
- Results of the system will be unreliable

Common Software Risks

- System will be difficult to use
- Programs will not be maintainable
- Systems will be able to interconnect with other computer systems
- Performance level will be unacceptable

Why are Defects Hard to Find

- Not Looking
  - Test often are not performed because a particular test condition was unknown.
  - Some parts of a system go untested because developers assume software changes don’t affect them
- Looking, But Not Seeing
  - Losing your car keys only to discover they were in plain sight the entire time

Levels of Testing

- Verification Testing (static testing development process)
- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing

Testing Techniques

- Structural versus Functional
- Verification versus Validation
- Static versus Dynamic Testing
- Examples of Specific Testing Techniques

Structural versus Functional

- Both structural and functional analysis should be performed to ensure adequate testing
- Structural tend to uncover errors during coding
- Functional is not concerned with how processing occurs, but with the results of processing

Structural Testing Techniques

- Stress
- Execution
- Recovery
- Operations
- Compliance (To Process)
- Security

Stress Testing

- Determine system performs with expected volumes
  - Normal or above-normal volumes of transaction
  - Structurally able to process large volumes of data
  - Sufficient resources available to meet expectations
  - User perform their tasks in desired turnaround time

Execution Technique

- System achieves desired level of proficiency
  - Performance level of the system structure.
  - Optimum use of hardware and software
  - Response time to online user requests
  - Transaction processing turnaround time

Recovery Testing

- System can be returned to an operational status after a failure
  - Evaluate process, procedures and documentation
  - Estimate potential loss time spans and backup recovery
  - Simulate Disaster

Operations Testing

- System can be executed in a normal operational status
  - Completeness of computer operator documentation
  - Ensure necessary support, such as job control are prepared and functional
  - Completeness of operator training
  - Ensure prepared documentation can, in fact, operate the system

Compliance

- System is developed in accordance with standards and procedures
  - System development and maintenance methodologies are followed
  - Department standards, procedures and guidelines
  - Completeness and reasonable of application system documentation

Security Testing

- System is protected in accordance with importance to organization
  - Resources are protected and access levels are defined
  - Security procedures have been implemented and function in accordance with the specifications.
  - System can identify and prevent unauthorized access

Functional Testing Techniques

- Requirements
- Regression
- Error Handling
- Manual Support
- Intersystem
- Control
- Parallel

Requirements

- System performs as specified
  - Proven system requirements
  - Compliance to policies and regulations
  - Maintain correctness over extended processing periods
  - Record retention

Regression

- Verifies that anything unchanged still performs correctly
  - Unchanged system segments function
  - Unchanged manual procedures correct
  - Focus on areas of exchange between new functionality and old
  - Very time-consuming and tedious operation

Error Handling

- Errors can be prevented or detected and then corrected
  - Requires to think negatively
  - Expected error conditions are handled by the application
  - Errors are properly handled
  - Reasonable controls are maintained during correction process

Manual Support

- The people-computer interaction works
  - Support procedures are documented and complete
  - Support staff are adequately trained
  - Support tools are implemented
  - Automated segment are properly interfaced

Intersystem

- Data is correctly passed from system to system
  - Proper parameters have been set
  - Data correctly passes between applications
  - Coordination and timing functions exists between systems
  - Systems are accurate and complete

Control

- Controls reduce system risk to an acceptable level
  - Data validation
  - File Integrity
  - Audit Trail
  - Backup and recovery
  - Transactions are authorized
  - Process is efficient, effective and economical

Parallel

- Old system and new system are run and the results compared to detect unplanned differences
  - System Calculations
  - Data processing
  - Ensure scheduled background tasks
  - Ensure operational status

Verification versus Validation

- Verification ensures the system complies with the organization' standards and processes. Did we build the right system?

- Validation physically ensures the system operates according to plan. Did we build the system right?

System Verification Examples

- Requirements Review
- Design Review
- Code Walkthroughs
- Code Inspections

System Validation Examples

- Unit Testing
- Integration Testing
- System Testing
- User Acceptance Testing

Static versus Dynamic Testing

- Static testing is performed using the software documentation. The code is not executing during static testing.
- Dynamic testing requires the code to be in an executable state to perform the tests.

Static Testing

- Verification techniques are static tests
  - Feasibility Reviews
  - Requirement Reviews
  - Design Reviews

Dynamic Testing

- Validation tests are dynamic tests
  - Unit Testing
  - Integration Testing
  - System Testing
  - User Acceptance

Examples of Testing Techniques

- White-Box Testing
- Black-Box Testing
- Incremental Testing
- Thread Testing
- Requirements Tracing
- Boundary Value Analysis
- Error Guessing and Special Value Analysis
- Cause-Effect Graphing
- Design-Based Functional Testing
- Coverage-Based Testing
- Complexity-Based Testing
- Statistical Analyses and Error Seeding
- Mutation Analysis
- Flow Analysis

White-Box Testing

- Assumes the path of logic in a unit or program is know.
- Consists of testing paths, branch by branch, to produce predictable results.
- Examples: Statement Coverage, Decision Coverage and Multiple Condition Coverage

Black-Box Testing

- Focuses on testing the function of the program or application against its specifications
- Often called Functional Testing
- Determines whether combinations of inputs and operations product expected results

Incremental Testing

- Discipline method of testing the interfaces between unit tested programs
- Top Down
  - Use interim stubs to simulate lower interfacing modules
- Bottom-Up
  - Drives to provide test input that call module or programs

Thread Testing

- Demonstrates key functional capabilities by testing a string of units that accomplish a specific function in the application
- Thread testing and incremental testing are usually utilized together

Requirements Tracing

- Primary goal of software testing is to prove requirements are delivered in the final product
- Trace functional and non-functional requirements through out analysis and design models, class and sequence diagrams, code and testing

Boundary Value Analysis

- Technique that consists of developing test cases and data that focus on the input and output boundaries of a given function
  - Low boundary +/- one ($9,999 and $10,001)
  - On the boundary ($10,000 and $15,000)
  - Upper boundary +/- one ($14,999 and $15,001)

Error Guessing and Special Value Analysis

- Certain test data seem highly probable to catch errors
  - February 29, 2000
  - Zero inputs
  - Negative value inputs
  - Special Characters
  - Data Type Values
  - Randomized selections

Cause-Effect Graphing

- Technique for developing test cases for programs form the high-level specifications
- Limited entry decision table is derived
  - Examples: Causes in 2 and 3 result in effect 4 or Causes in 2, 3 and 5 result in 6

Design-Based Functional Testing

- Top Down Approach (Functional Design Tree)
- Necessary to invent other smaller functions from requirements
- A requirement become the root node
- Corresponding design functions become the second level of the tree

Coverage-Based Testing

- Top down design
- Knowledge and access to code base
- Number of statements, branches or paths in the program
- Test data should cause execution of all paths

Complexity-Based Testing

- McCabe metrics
  - Cyclomatic
  - Essential
  - Actual
- Cyclomatic and Essential are calculated from a program graph; linearly independent program paths
- Actual is a runtime metric

Statistical Analyses and Error Seeding

- Most common type of test data analysis
- Insert known errors in the code
- Execute test data and determine number of actual errors

Mutation Analysis

- Derived from branch coverage and statement coverage
- Seed the program to be tested with errors, creating several mutants of the original problem
- Determines how deep will the error pass through the code

Flow Analysis

- Control flow
  - Analyze program behavior
  - Locate instrumentation breakpoints
  - Identify paths
- Data flow
  - Discover program anomalies such as undefined or unreferenced variables

Combining Testing Techniques

- Forms more powerful and efficient testing technique
- Merge standard testing techniques with formal verification
- Tradeoffs between testing and formal methods are efficient and effective

Questions?