

SOFTWARE TESTING

Dr. KOYEL DATTA GUPTA

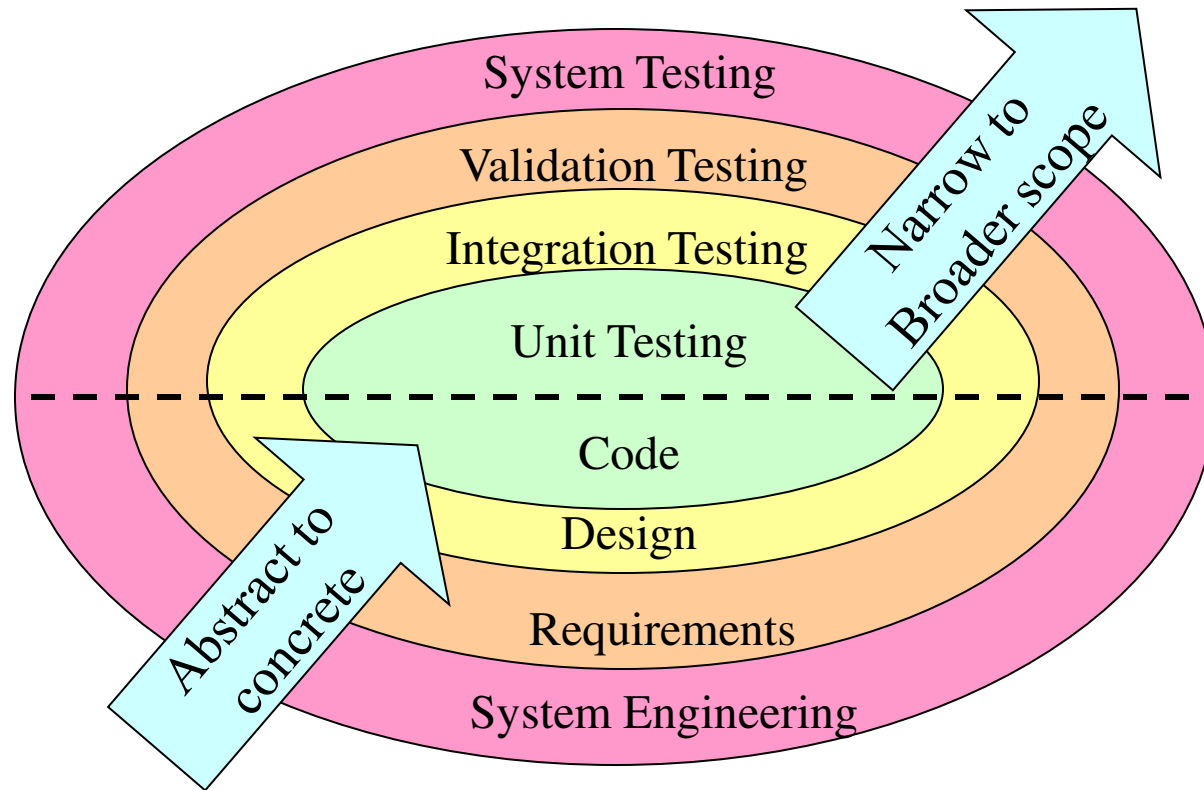
OUTLINE

- INTRODUCTION
- A STRATEGY FOR TESTING CONVENTIONAL SOFTWARE
- LEVELS OF TESTING FOR CONVENTIONAL SOFTWARE
- TESTING STRATEGY APPLIED TO CONVENTIONAL SOFTWARE
- UNIT TESTING
- COMMON COMPUTATIONAL ERRORS IN EXECUTION PATHS
- INTEGRATION TESTING
- REGRESSION TESTING
- SMOKE TESTING
- VALIDATION TESTING
- ALPHA AND BETA TESTING
- SYSTEM TESTING

INTRODUCTION

- A strategy for software testing integrates the design of software test cases into a well-planned series of steps that result in successful development of the software
- The strategy provides a road map that describes the steps to be taken, when, and how much effort, time, and resources will be required
- The strategy incorporates test planning, test case design, test execution, and test result collection and evaluation
- The strategy provides guidance for the practitioner and a set of milestones for the manager
- Because of time pressures, progress must be measurable and problems must surface as early as possible

A STRATEGY FOR TESTING CONVENTIONAL SOFTWARE



LEVELS OF TESTING FOR CONVENTIONAL SOFTWARE

- Unit testing
 - Concentrates on each component/function of the software as implemented in the source code
- Integration testing
 - Focuses on the design and construction of the software architecture
- Validation testing
 - Requirements are validated against the constructed software
- System testing
 - The software and other system elements are tested as a whole

TESTING STRATEGY APPLIED TO CONVENTIONAL SOFTWARE

- Unit testing
 - Exercises specific paths in a component's control structure to ensure complete coverage and maximum error detection
 - Components are then assembled and integrated
- Integration testing
 - Focuses on inputs and outputs, and how well the components fit together and work together
- Validation testing
 - Provides final assurance that the software meets all functional, behavioral, and performance requirements
- System testing
 - Verifies that all system elements (software, hardware, people, databases) mesh properly and that overall system function and performance is achieved

TESTING STRATEGY APPLIED TO OBJECT-ORIENTED SOFTWARE

- Must broaden testing to include detections of errors in analysis and design models
- Unit testing loses some of its meaning and integration testing changes significantly
- Use the same philosophy but different approach as in conventional software testing
- Test "in the small" and then work out to testing "in the large"
 - Testing in the small involves class attributes and operations; the main focus is on communication and collaboration within the class
 - Testing in the large involves a series of regression tests to uncover errors due to communication and collaboration among classes
- Finally, the system as a whole is tested to detect errors in fulfilling requirements

UNIT TESTING

- Focuses testing on the function or software module
- Concentrates on the internal processing logic and data structures
- Is simplified when a module is designed with high cohesion
 - Reduces the number of test cases
 - Allows errors to be more easily predicted and uncovered
- Concentrates on critical modules and those with **high cyclomatic complexity** when testing resources are limited

TARGETS FOR UNIT TEST CASES

- Module interface
 - Ensure that information flows properly into and out of the module
- Local data structures
 - Ensure that data stored temporarily maintains its integrity during all steps in an algorithm execution
- Boundary conditions
 - Ensure that the module operates properly at boundary values established to limit or restrict processing
- Independent paths (basis paths)
 - Paths are exercised to ensure that all statements in a module have been executed at least once
- Error handling paths
 - Ensure that the algorithms respond correctly to specific error conditions

COMMON COMPUTATIONAL ERRORS IN EXECUTION PATHS

- Misunderstood or incorrect arithmetic precedence
- Mixed mode operations (e.g., int, float, char)
- Incorrect initialization of values
- Precision inaccuracy and round-off errors
- Incorrect symbolic representation of an expression (int vs. float)

OTHER ERRORS TO UNCOVER

- Comparison of different data types
- Incorrect logical operators or precedence
- Expectation of equality when precision error makes equality unlikely (using == with float types)
- Incorrect comparison of variables
- Improper or nonexistent loop termination
- Failure to exit when divergent iteration is encountered
- Improperly modified loop variables
- Boundary value violations

PROBLEMS TO UNCOVER IN ERROR HANDLING

- Error description is unintelligible or ambiguous
- Error noted does not correspond to error encountered
- Error condition causes operating system intervention prior to error handling
- Exception condition processing is incorrect
- Error description does not provide enough information to assist in the location of the cause of the error

DRIVERS AND STUBS FOR UNIT TESTING

- Driver
 - A simple main program that accepts test case data, passes such data to the component being tested, and prints the returned results
- Stubs
 - Serve to replace modules that are subordinate to (called by) the component to be tested
 - It uses the module's exact interface, may do minimal data manipulation, provides verification of entry, and returns control to the module undergoing testing
- Drivers and stubs both represent overhead
 - Both must be written but don't constitute part of the installed software product

INTEGRATION TESTING

- Defined as a systematic technique for constructing the software architecture
 - At the same time integration is occurring, conduct tests to uncover errors associated with interfaces
- Objective is to take unit tested modules and build a program structure based on the prescribed design
- Two Approaches
 - Non-incremental Integration Testing
 - Incremental Integration Testing

NON-INCREMENTAL INTEGRATION TESTING

- Commonly called the “Big Bang” approach
- All components are combined in advance
- The entire program is tested as a whole
- Chaos results
- Many seemingly-unrelated errors are encountered
- Correction is difficult because isolation of causes is complicated
- Once a set of errors are corrected, more errors occur, and testing appears to enter an endless loop

INCREMENTAL INTEGRATION TESTING

- Three kinds
 - Top-down integration
 - Bottom-up integration
 - Sandwich integration
- The program is constructed and tested in small increments
- Errors are easier to isolate and correct
- Interfaces are more likely to be tested completely
- A systematic test approach is applied

TOP-DOWN INTEGRATION

- Modules are integrated by moving downward through the control hierarchy, beginning with the main module
- Subordinate modules are incorporated in either a depth-first or breadth-first fashion
 - DF: All modules on a major control path are integrated
 - BF: All modules directly subordinate at each level are integrated
- Advantages
 - This approach verifies major control or decision points early in the test process
- Disadvantages
 - Stubs need to be created to substitute for modules that have not been built or tested yet; this code is later discarded
 - Because stubs are used to replace lower level modules, no significant data flow can occur until much later in the integration/testing process

BOTTOM-UP INTEGRATION

- Integration and testing starts with the most atomic modules in the control hierarchy
- Advantages
 - This approach verifies low-level data processing early in the testing process
 - Need for stubs is eliminated
- Disadvantages
 - Driver modules need to be built to test the lower-level modules; this code is later discarded or expanded into a full-featured version
 - Drivers inherently do not contain the complete algorithms that will eventually use the services of the lower-level modules; consequently, testing may be incomplete or more testing may be needed later when the upper level modules are available

SANDWICH INTEGRATION

- Consists of a combination of both top-down and bottom-up integration
- Occurs both at the highest level modules and also at the lowest level modules
- Proceeds using functional groups of modules, with each group completed before the next
 - High and low-level modules are grouped based on the control and data processing they provide for a specific program feature
 - Integration within the group progresses in alternating steps between the high and low level modules of the group
 - When integration for a certain functional group is complete, integration and testing moves onto the next group
- Reaps the advantages of both types of integration while minimizing the need for drivers and stubs
- Requires a disciplined approach so that integration doesn't tend towards the "big bang" scenario

REGRESSION TESTING

- Each new addition or change to baselined software may cause problems with functions that previously worked flawlessly
- Regression testing re-executes a small subset of tests that have already been conducted
 - Ensures that changes have not propagated unintended side effects
 - Helps to ensure that changes do not introduce unintended behavior or additional errors
 - May be done manually or through the use of automated capture/playback tools
- Regression test suite contains three different classes of test cases
 - A representative sample of tests that will exercise all software functions
 - Additional tests that focus on software functions that are likely to be affected by the change
 - Tests that focus on the actual software components that have been changed

SMOKE TESTING

- Taken from the world of hardware
 - Power is applied and a technician checks for sparks, smoke, or other dramatic signs of fundamental failure
- Designed as a pacing mechanism for time-critical projects
 - Allows the software team to assess its project on a frequent basis
- Includes the following activities
 - The software is compiled and linked into a build
 - A series of breadth tests is designed to expose errors that will keep the build from properly performing its function
 - The goal is to uncover “show stopper” errors that have the highest likelihood of throwing the software project behind schedule
 - The build is integrated with other builds and the entire product is smoke tested daily
 - Daily testing gives managers and practitioners a realistic assessment of the progress of the integration testing
 - After a smoke test is completed, detailed test scripts are executed

BENEFITS OF SMOKE TESTING

- Integration risk is minimized
 - Daily testing uncovers incompatibilities and show-stoppers early in the testing process, thereby reducing schedule impact
- The quality of the end-product is improved
 - Smoke testing is likely to uncover both functional errors and architectural and component-level design errors
- Error diagnosis and correction are simplified
 - Smoke testing will probably uncover errors in the newest components that were integrated
- Progress is easier to assess
 - As integration testing progresses, more software has been integrated and more has been demonstrated to work
 - Managers get a good indication that progress is being made

TEST STRATEGIES FOR OBJECT-ORIENTED SOFTWARE

- With object-oriented software, you can no longer test a single operation in isolation (conventional thinking)
- Traditional top-down or bottom-up integration testing has little meaning
- Class testing for object-oriented software is the equivalent of unit testing for conventional software
 - Focuses on operations encapsulated by the class and the state behavior of the class
- Drivers can be used
 - To test operations at the lowest level and for testing whole groups of classes
 - To replace the user interface so that tests of system functionality can be conducted prior to implementation of the actual interface
- Stubs can be used
 - In situations in which collaboration between classes is required but one or more of the collaborating classes has not yet been fully implemented

TEST STRATEGIES FOR OBJECT-ORIENTED SOFTWARE (CONTINUED)

- Two different object-oriented testing strategies
 - Thread-based testing
 - Integrates the set of classes required to respond to one input or event for the system
 - Each thread is integrated and tested individually
 - Regression testing is applied to ensure that no side effects occur
 - Use-based testing
 - First tests the independent classes that use very few, if any, server classes
 - Then the next layer of classes, called dependent classes, are integrated
 - This sequence of testing layer of dependent classes continues until the entire system is constructed

VALIDATION TESTING

- Validation testing follows integration testing
- The distinction between conventional and object-oriented software disappears
- Focuses on user-visible actions and user-recognizable output from the system
- Demonstrates conformity with requirements
- Designed to ensure that
 - All functional requirements are satisfied
 - All behavioral characteristics are achieved
 - All performance requirements are attained
 - Documentation is correct
 - Usability and other requirements are met (e.g., transportability, compatibility, error recovery, maintainability)
- After each validation test
 - The function or performance characteristic conforms to specification and is accepted
 - A deviation from specification is uncovered and a deficiency list is created
- A configuration review or audit ensures that all elements of the software configuration have been properly developed, cataloged, and have the necessary detail for entering the support phase of the software life cycle

ALPHA AND BETA TESTING

- Alpha testing
 - Conducted at the developer's site by end users
 - Software is used in a natural setting with developers watching intently
 - Testing is conducted in a controlled environment
- Beta testing
 - Conducted at end-user sites
 - Developer is generally not present
 - It serves as a live application of the software in an environment that cannot be controlled by the developer
 - The end-user records all problems that are encountered and reports these to the developers at regular intervals
- After beta testing is complete, software engineers make software modifications and prepare for release of the software product to the entire customer base

SYSTEM TESTING

- Recovery testing
 - Tests for recovery from system faults
 - Forces the software to fail in a variety of ways and verifies that recovery is properly performed
 - Tests reinitialization, checkpointing mechanisms, data recovery, and restart for correctness
- Security testing
 - Verifies that protection mechanisms built into a system will, in fact, protect it from improper access
- Stress testing
 - Executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
- Performance testing
 - Tests the run-time performance of software within the context of an integrated system
 - Often coupled with stress testing and usually requires both hardware and software instrumentation
 - Can uncover situations that lead to degradation and possible system failure