Chapter 19: Software Testing—Component Level

February 22, 2021 9:00 AM

- Strategic Approach to Testing
  - Conduct effective technical reviews before testing begins
  - Testing begins at the component level and works toward integration on the entire system
  - Use different testing techniques for the appropriate software
  - Testing is conducted by the developer of the software and an independent test group
  - Testing and debugging are different, debugging must be used in testing
- Verification and Validation
  - ★ Verification ensures that software correctly implements a function
  - ★ Validation ensures that software is traceable to customer requirements
- Organizing for Testing
  - Software developers are responsible for testing individual program components
  - When software architecture is complete then the independent test group is involved
  - ★ The independent test group (ITG) is there to prevent the builder from testing their own product
  - ITG personnel are paid to find errors
  - Developers and ITG work closely to ensure thorough tests are conducted
- Testing Strategy - figure picture
  - System testing
  - Validation testing
  - Integration testing
- Role of Scaffolding
  - ★ Scaffolding is required to create a testing framework
  - A driver must be developed for each unit test
  - A driver is a "main program" that accepts testcase data
  - Stubs (dummy subprogram) replace modules invoked by the component to be tested
  - A stub uses the module's interface, may do minimal manipulation, prints verification entry, and returns control to the module undergoing testing
- Criteria for Done
  - Testing is never done; the burden is shifted from the engineer to the user (wrong)
  - You're done testing when you are out of time or money (wrong)
  - The statistical quality assurance approach suggests executing tests derived from a statistical sample of all possible program executions by all targeted users
  - ★ By collecting metrics during testing and making use of existing statistical models, it is possible to develop meaningful guidelines for answering the question: "When are we done testing?"
- Test Planning
  - Specify quantifiable measures of the requirements before testing commences
  - State testing objectives explicitly
  - Understand the users of the software and develop a profile for each user category
  - Develop a testing plan that emphasizes "rapid cycle testing"
  - ★ Rapid cycle testing tests at the end of every sprint
  - Build a robust software that is designed to test itself
  - Use effective technical reviews as a filter prior to testing
  - Conduct technical reviews to assess the strategy and test cases themselves
  - Develop a continuous improvement approach for the testing process
- Test Recordkeeping
  - Briefly describes the test case

    - Have a pointer to the requirement being tested
    - Have expected output from the test case data on the criteria for success
    - Indicate whether the test was passed or failed
    - Dates the test case was run
    - Should have room for comments about why a test may have failed (aids in debugging)
- Cost Effective Testing
    - Exhaustive testing requires every possible combination and ordering of input values be processed by the test component
    - The return on exhaustive testing is often not worth the effort
    - Testers should work smarter and allocate their testing resources on modules crucial to the success of the project or those that are suspected to be error-prone as the focus of their testing unit
- Test Case Design
    - Design unit test cases before you develop code for a component to ensure that code will pass the tests
    - Test cases are designed to cover the following areas:
        - ★ The module interface is tested to ensure that information properly flows into and out of the program unit.
        - ★ Local data structures are examined to ensure that stored data maintains its integrity during execution
        - ★ Independent paths through control structures are exercised to ensure all statements are executed at least once
        - ★ Boundary conditions are tested to ensure module operates properly at boundaries established to limit or restrict processing
        - ★ All error-handling paths are tested
- What is a "Good" Test?
    - A good test has a high probability of finding an error
    - A good test is not redundant
    - A good test should be "best of breed"
    - A good test should be neither too simple nor too complex
- Error Handling
    - A good design anticipates error conditions and establishes error-handling paths which must be tested
    - Among the potential errors that should be tested when error handling is evaluated are:
        - Error description is unintelligible
        - Error noted does not correspond to error encountered
        - Error condition causes system intervention prior to error handling
        - Exception-condition processing is incorrect
        - Error description does not give enough information
- Traceability
    - ★ To ensure that the testing process is auditable, each test case needs to be traceable back to specific functional or non-functional requirements to anti-requirements
    - Often non-functional requirements need to be traceable to specific business or architectural requirements
    - Many test process failures can be traced to missing traceability paths, inconsistent test data, or incomplete test coverage
    - ★ Regression testing requires retesting of selected components that may be affected by changes made to other collaborating software components
- ★ White Box Testing (Glass Box Testing, Clear Box Testing)
    - Using white-box testing methods, you can derive test cases that:
        - Guarantee that all independent paths within a module have been exercised at least once
        - Exercise all logical decisions on their true and false sides

        - Execute all loops at their boundaries and within their operational bounds
        - Exercise internal data structures to ensure validity
- Basic Path Testing
    - Determines the number of independent paths in the program by computing Complexity:
        - ★ The number of regions of the flow graph corresponds to the cyclomatic complexity (book example has 4)
        - ★ Cyclomatic complexity $V(G)$ for a flow graph G is defined as (E = Edge, N = Node, P = Predicate Node)
            - □ $V(G) = E - N + 2$
            - □ $V(G) = P + 1$
        - ★ An independent path is any path through the program that introduces at least one new set of processing statements or a new condition (book examples)
            - □ Path 1: 1-11
            - □ Path 2: 1-2,3-4,5-10-1-11
            - □ Path 3: 1-2-3-6-8-9-10-1-11
            - □ Path 4: 1-2-3-6-7-9-10-1-11
    - Designing test cases
        - Use the code as a foundation
        - Determine the cyclomatic complexity of the flow graph
        - Determine a basis set of linearly independent paths
        - Prepare the test cases that will force execution of each path in the basis set
- Control Structure Testing
    - ★ Condition testing is a test-case design method that exercises logical conditions contained in a program module
    - ★ Data flow testing selects test paths according to the locations of definitions and uses variables in the program
    - ★ Loop testing is a white-box testing technique that focuses exclusively on the validity of loop constructs
- Loop Testing
    - Test cases for simple loops:
        - Skip the loop entirely
        - One pass through the loop
        - Two passes through the loop
        - m passes through the loop where m < n (normally)
        - n - 1, n, n + 1 passes through the loop (boundaries)
    - Test cases for nested loops:
        - Start at the innermost loop and set all other loops to minimum values
        - Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter values
        - Add other tests for excluded values
        - Work outward, conducting tests for the next loop, but keeping all the other outer loops at minimum values and other nested loops to "typical" values
        - Continue until all loops have been tested
- ★ Black Box Testing
    - Black-box (functional) testing attempts to find errors in the following categories:
        - Incorrect or missing functions
        - Interface errors
        - Errors in data structures or external database access
        - Behavior or performance issues
        - Initialization and termination errors
    - Questions
        - How is functional validity tested?
        - How are system behavior and performance tested?

- What classes of input will make a good test case?
- Is the system particularly sensitive to certain input values?
- Interface Testing
  - ★ Interface testing is used to check that a program component accepts information passed to it in the proper order and data types and returns information in proper order and data format
  - Components are not stand-alone programs testing interfaces requires the use of stubs and drivers
  - Stubs and drivers sometimes incorporate test cases to be passed to the component or accessed by the component

- Object-Oriented Testing (OOT)
  - To adequately test OO systems, three things must be done:
    - The definition of testing must be broadened to include error discovery techniques applied to object-oriented analysis and design models
    - The strategy for unit and integration testing must change significantly
    - The design of test cases must account for the unique characteristics of OO software
  - Class Testing (Unit testing)
    - Class testing for OO software is the equivalent of unit testing for conventional software
    - Unlike unit testing of conventional software, which tends to focus on the algorithmic detail of a module and the data that flows across the module interface
    - Class testing for OO software is driven by the operations encapsulated by the class and the state behavior of the class
  - Behavior Testing
    - ★ A state diagram can be used to help derive a sequence of tests that will exercise dynamic behavior of the class
    - Tests to be designed should achieve full coverage by using operation sequences cause transitions through all allowable states
    - When class behavior results in a collaboration with several classes, multiple state diagrams can be used to track system behavioral flow
  - Boundary Value Analysis (BVA)
    - ★ Boundary value analysis leads to a selection of test cases that exercise bounding values
    - Guidelines for BVA:
      - If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b just above and just below a and b
      - If an input condition specifies a number of values, test cases should be developed that exercise the min and max numbers as well as values just above and below min and max
      - Apply guidelines 1 and 2 to output conditions
      - If internal program data structures have prescribed boundaries be certain to design a test case to exercise the data structure at its boundary