Software Testing

- Complex process
- Goals
  - Exercise a program
  - To find errors prior to delivery
- Verification
  - Does it work right?
- Validation
  - Does it do the right thing?

Testability (1)

- Operability
  - It operates cleanly
- Observability
  - Results of each test case are readily observed
- Controlability
  - Can we automate and optimize testing
- Decomposability
  - Testing can be targeted

Testability (2)

- Simplicity
  - Reduce complex architecture and logic to simplify tests
- Stability
  - Few changes are requested during testing
- Understandability
  - Of the design

What Testing Shows

- Errors
  - But never their absence
- Conformance
  - To the requirements
- Performance
  - Speed, etc.
- Quality
  - But only an indicator of it

Who Tests the Software?

- Developer
  - Understands the system
  - May test gently
  - Takes errors too personally
  - Concern may be deadlines
- Independent tester
  - Takes longer to learn system
  - Concern is quality

Test Case Design

- Objective
  - To uncover errors
- Criteria
  - In a complete manner
- Constraint
  - In a timely and efficient manner
- Problem
  - Bugs usually hide

White Box Testing (1)

- Details of the implementation are known
  - Study the code
- Purpose
  - Test all execution paths
  - Test all boundary conditions
    - Min, max, a few in the middle, out of range, etc.

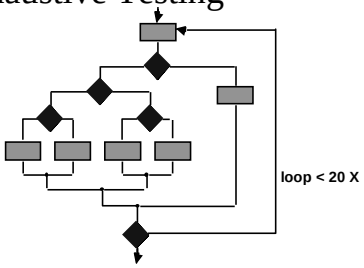
White Box Testing (2)

- Number of paths subject to combinational explosion
- Ex: Loop around switch-case (5), and if (3)
  - #paths = $(5*3)^{\text{loop count}}$
- All possible paths not practical
- Most paths have dependencies
  - Identify the truly independent paths

White Box Testing (3)

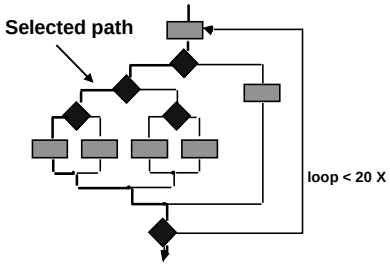
- If statements
  - Take special care with < vs. ≤ etc.
- Else, else-if clauses
  - Is correct sub-choice taken (watch order)
- Switch-case statements
  - default (?)
- Loops
  - Skip, one pass, two passes, too many passes

Exhaustive Testing



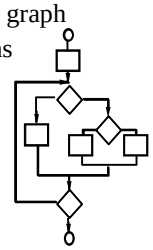
There are 10^{14} possible paths (5^{20})!
If we execute one test per millisecond,
it would take 3,170 years to test this program!!

Selective Testing



Cyclomatic Complexity

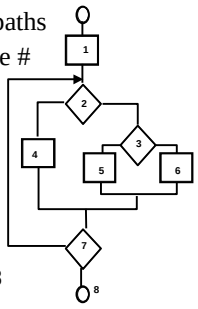
- Quantitative measure: V(G)
  - Number of regions in a flow graph
  - Number of independent paths
- Computing it
  - #Edges – #Nodes + 2 -or-
  - #Predicates(decisions) + 1
- Large V(G)
  - Greater likelihood of errors



Basis Path Testing

1. Derive the independent paths
   - Use V(G) to determine #
2. Derive test cases
   - To exercise each path

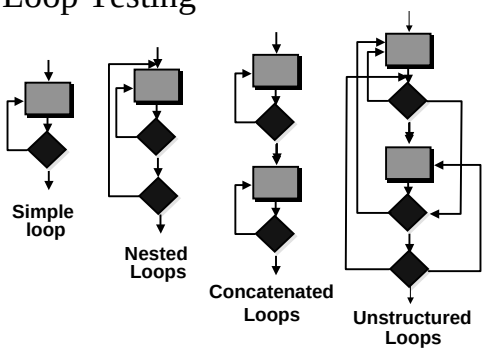
Example:
Path1: 1,2,3,6,7,8
Path2: 1,2,3,5,7,8
Path3: 1,2,4,7,8
Path4: 1,2,7,2,4,7,1,…,7,8



Basis Path Testing Notes

- A flow chart/graph isn't necessary
  - But it can be very helpful
- Compound tests
  - Count as two or more paths
- Apply to **ALL** critical modules

Loop Testing



Loop Testing: Simple Loops

- No passes through the loop
- One pass through the loop
- Two passes through the loop
- m passes through the loop (m < max)
- max-1, max passes through the loop
- Try to force m+1 passes

Loop Testing: Nested Loops

- Test the innermost loop
  - Set all outer loops to minimum
  - Test min, min+1, max-1, max, and others
- Move out one loop
  - Set inner loops to typical
  - Outer loops to max
- Continue until the outer loop is done

Loop Testing: Others

- Concatenated
  - Independent loops
    - Treat each loop as a simple loop
  - Dependent loops
    - Treat as if nested
- Unstructured
  - Have fun!

Black Box Testing (1)

- Don't know the insides
- Only know the interface and specification
- Inputs
  - Look at the range specifications
    - In range
    - Range boundaries
    - Outside the range

Black Box Testing (2)

- Outputs
  - Try to apply inputs the produce outputs at the boundary
- Comparison
  - Validate against other implementations
    - Verify values and behavior

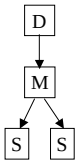
Error Testing

- Try to cause all "detected" error conditions
  - Syntax, invalid values, missing files, etc.
- Evaluate the response
  - Are the error messages appropriate?
  - Did the system recover acceptably?
  - Are all the errors detected?

Testing Strategy

- Unit test
  - Individual modules and functions
- Integration test
  - Combining modules
- System test
  - All of the modules
- Validation test
  - Conformance to requirements

Unit Testing

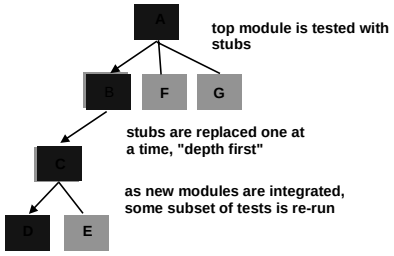


- One unit/class at a time
- Provide a testing scaffold
  - Drivers – layers above
    - Using white/black box ideas
    - Interfaces
  - Stubs – layers below
    - Sophisticated enough to allow all paths

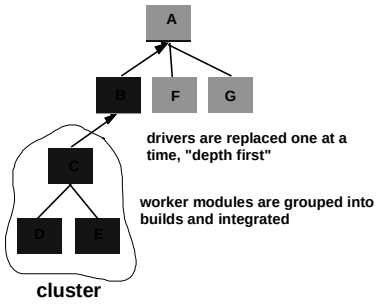
Integration Testing

- Direction of integration
  - From bottom-top
    - Replace stubs
  - From top-bottom
    - Replace drivers
  - Sandwich
- Level of integration
  - Incremental
  - All at once

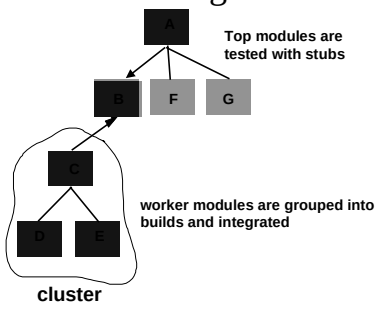
Top Down Integration



Bottom-Up Integration

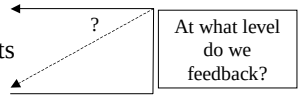


Sandwich Testing



Testing Iterations

- How thorough do we want to be?
- Typical approach
  - Build units
  - Test units
  - Integrate units
  - Test system
- Do we run every test every iteration?



Iteration Frequency

- Short duration (daily)
  - Provides timely feedback
  - May give poor feedback
    - Due to unready units
- Long duration (weekly, erratic?)
  - Delays feedback
  - Perhaps too late to meet a deadline

Automated Build and Test

- Developers check in code
  - Per QA procedures
- System build and test
  - Overnight
    - Driven by script
- Validation
  - First thing

Regression Testing

- Prevent back sliding
  - Don't want old errors to come back
- When we find a defect
  - Design a specific test for it
  - Verify test, fix the defect, retest
  - Add test to the test script

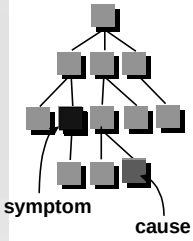
Validation Test

- User/client/customer driven tests
  - Provide validation
- Alpha test
  - Before we put the system in use
  - Typically at the developer's site
- Beta test
  - At customer site ("beta site")

The Debugging Process

- Test cases
  - Uncover error
- Cause unknown
  - Develop experiments
- Experiments
  - Reveal source
- Regression tests?

Symptoms & Causes



- Geometric separation
- One error may mask another
- Source may be compound
- System or compiler is the source
- Erroneous assumptions
- Intermittent

Debugging Advice

- Be systematic
- Use tools
  - Debuggers, etc.
- Get help
  - Fresh insight is amazing
- Regression test
  - Don’t let it burn you again

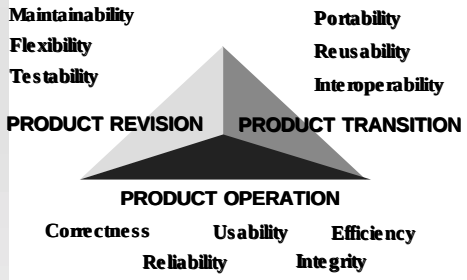
When are We Done?

- Time limit?
- Cost limit?
- New error frequency falls?
- Statistical models exist
  - Predict number of remaining defects
  - # of defects expected vs. additional resources (time and money)

Test Plan

- Written testing sequence – thorough, early
  - User input
    - Prompts, GUI order
  - Files, etc.
    - Sample files, system data
  - Expected output
    - Pass/fail criteria

McCall's Triangle of Quality



Measurement Principles

- Establish objectives
  - Before data collection begins
- Be unambiguous
- Based on theory
  - Demonstrates applicability
- Customized
  - For each domain or product

Collection and Analysis Principles

- Automate
  - Whenever possible
- Use valid statistical techniques
  - Determine relationship between metrics and quality
- Establish interpretive guidelines
- Consider objective over subjective

Attributes (1)

- Simple and computable
  - Easy to derive and compute
  - Don't overly load your process
- Empirically and intuitively persuasive
  - Satisfy notion of what is being measured
- Consistent and objective
  - Result should be unambiguous
  - Don't combine disparate measures

Attributes (2)

- Language independent
  - Tied to our models, not the program
    - Analysis and/or design
- Effective for quality feedback
  - Provide valuable information
  - That can be used

Analysis Metrics

- Function-based metrics
  - Function and feature points
- Bang metric
  - Estimates size
  - Combines data, functional and behavioral models
- Specification metrics
  - Quality measure
  - Looks at number of requirements by type

Architectural Design Metrics

- Architectural design metrics
  - Structural = g(fan-out)
  - Data = f(I/O variables, fan-out)
  - System = h(structural & data complexity)
- HK metric
  - Function of fan-in and fan-out
- Morphology metrics
  - Number of modules and interfaces

Component-Level Design Metrics

- Cohesion metrics
  - Data objects
  - Area of their definition/scope
- Coupling metrics
  - I/O parameters, globals, & modules used
- Complexity metrics
  - Too many to enumerate

Interface Design Metrics

- Layout appropriateness
  - Positioning
  - Layout
  - Cost of “changing” views and data

Code Metrics

- Halstead’s Software Science
  - Collection of metrics
- Examines
  - Number and type of operators
  - Complexity of operands