Validation and Verification

• Validation – process of evaluating software at the end of its development to ensure that it is free from failure and complies with its requirements
  – Building the right product
• Verification – process of determining whether or not the product fulfills the requirements established at a previous phase
  – Building the product right

Limitations

• Can not test for all possible data
• Can not test all possible paths

Approaches

• Non-Execution Based QA –
  • technical reviews, walkthroughs, inspections and audits
• Execution Based -- Testing
  – Module/unit testing
  – Integration testing
  – Validation/system/acceptance testing
  – Regression testing
• Proofs of correctness

Planned Activity

• Manpower
• Machine power

Testing

• Positive activity designed to show that a program is correct and meets its specification
• Negative activity designed to expose the presence of errors

Debugging

• Process of removing program errors
  – Reactive process which stems from testing
Planning Testing
• Definable
• How and what is to be tested must be specified at each stage of the cycle
• Specifications
• Modules content
• Inter-module connections

Test case design methods
• Black box
  – Test cases demonstrate that each specified function is fully operational
  – No regard to internal structure
• White/clear box
  – Test cases which assure that the internal operational components have been exercised

White/Clear Box
• Use control structures to derive test cases
• Guarantee all independent paths have been exercised at least once
• Exercise all logicals on both true and false
• Execute loops at boundaries and within
• Exercise all internal data structures

Black Box
• Focuses on functional requirements
• Derive a set of input conditions that fully exercise all functional requirements
• Finds
  – Incorrect or missing functions
  – Interface errors
  – Data structure of external data access errors
  – Performance errors
  – Initialization/termination errors

Automated testing tools
• Test file generators
• Test data generators
• File comparers
Module/Unit testing
- White box oriented
- Conducted in parallel
- Higher level of cohesion the less unit testing needed
- Test
  - Module interface to assure information properly flows
  - Data structure to assure that data storage maintains integrity
  - Boundary conditions
    - Establish limits
  - Independent paths are executed at least once
  - Error-handling paths
- Common errors found
  - Incorrect arithmetic or operator precedence
  - Mixed mode operations
  - Incorrect initialization
  - Precision inaccuracy
  - Data type mismatch
  - Incorrect logical operator precedence
  - Expected equality when precision errors make that unlikely
  - Improper or non-existent loop termination
  - Improperly modified loop variable
  - Error handling procedure
    - Description is unintelligible
    - Error noted doesn’t correspond to error encountered
    - Error causes system intervention PRIOR to error handling
    - Exception handling is incorrect

Integration Testing
- Module does not stand alone, it interacts with others
- Driver – accepts test cases and passes the data to the module and then prints results returns from module
- Stub – replace subordinate modules with dummies that accept data passed and does the minimum manipulation and prints verification of entry and exit
- Drivers and stubs create overhead
- Integration – the way in which we put the modules together
  - Objective is to take the modules and put them together systematically to create the package design
  - Testing the interfaces
• Why need test interfaces
  – Loose data between modules
  – One module can adversely effect another
  – Sub-functions when put together may not create the overall function
  – Precision errors while acceptable in each unit when put together become unacceptable

**Basic Approaches to Integration Testing**
• Big bang
  – Test all interfaces at once
  – Difficulties arise
    • Isolating the cause of malfunction
    • One correction causes new errors
  – Handling logic and interfaces at once
  – Endless chaos
  – Expensive machine time – recompile whole each fix
• Incremental
  – Work in small segments systematically
  – Top-down
  – Bottom-up
  – Sandwich
  – Execution order
  – Critical path

**Top-Down**
• Downwards through the structure chart
• Breadth first → level
• Depth first → leg
• Start with main add one more each time conducting tests
• If any changes are made – regression testing
• Advantages
  – Verify major control and decision points early
• Problems
  – Use stubs so no sign if data flows upward

**Bottom-Up**
• Start with lowest levels and add upper levels one at a time
• Advantages
  – No stubs are ever needed
– As move up need fewer drivers

• Problems
  – Major control and decision points done last
  – Not an entity until end

sandwich
• Solves problems with top-down and bottom-up
• Do top 2 levels top-down and the rest bottom-up

Execution order
• do mental execution and use it to order the inclusion of modules
• Advantages
  – Data dependencies are limited
• Disadvantages
  – Limits parallel development

Critical Path
• Identify critical modules
  – Address several functions
  – High level of control
  – Complex or error prone
  – Performance constraints
• Looking at
  – Data dependencies
  – Control dependencies
  – Limit need for drivers and stubs
• Output modules – early
• Wrapping modules – early
• Exception handling modules – later
• In determining order
  – Start with mental execution and focus on control and data

Acceptance/Validation Testing
• Criteria was determined and agreed upon during requirements
• Black box tests
• Show functional and non-functional requirements are met
• Alpha
  – At developers site by customer with developer looking over the shoulder
• Beta
  – At end user site without the developer

**System Testing**
• Hard & software together with other hardware/software its interacts
• Recovery rates
  – Force failures and see what happens
  – Record events
    • Automatic or human intervention
    • How long etc
• Security
  – Check on protection by trying to break in
• Stress testing
  – How much it can withstand without failing
  – Demands on resources