Software Testing Strategies-Introduction and Unit Testing

Basic Steps for Testing

- Test Plan
- Test Case Design
- Test Execution
- Data Collection and Evaluation

Characteristics of Testing Strategy

- **Testing strategy** should be
  - Flexible
    - Support Customized Testing
  - Rigid
    - Properly Planned and Managed
- “Testing strategy should not be haphazard”
  - Wastes time
  - Errors remain uncovered

Testing – From Small to Large

- Testing begins “in the small” and progresses “to the large”
    - From single component to the entire system

- **Outcome of Testing Phase:**

    - Test Specification Document

Verification vs. Validation

- Software testing is one element of a broader topic that is often referred to as verification and validation (V&V).
- "Verification refers to the set of tasks that ensure that software correctly implements a specific function."
- "Validation refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements."
- Boehm states this another way:
  - Verification: "Are we building the product right?"
  - Validation: "Are we building the right product?"

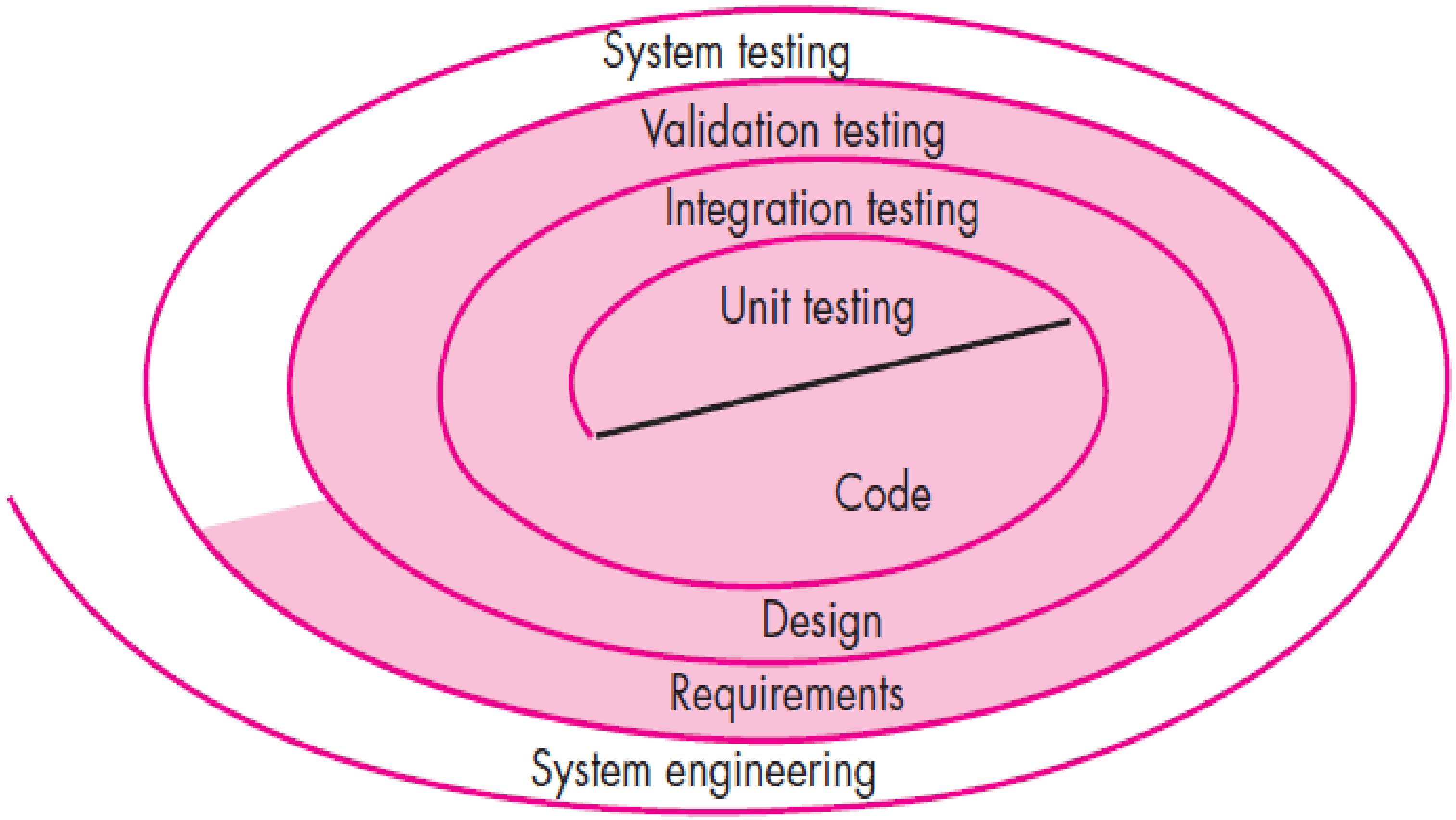
Who should perform Testing?

- Developers? – Can be biased
- Independent Test Group? – Not Enough

- Ideal way – Developers + ITG

- “Testers have to remain active during analysis and design phases also”

Testing – The Big Picture



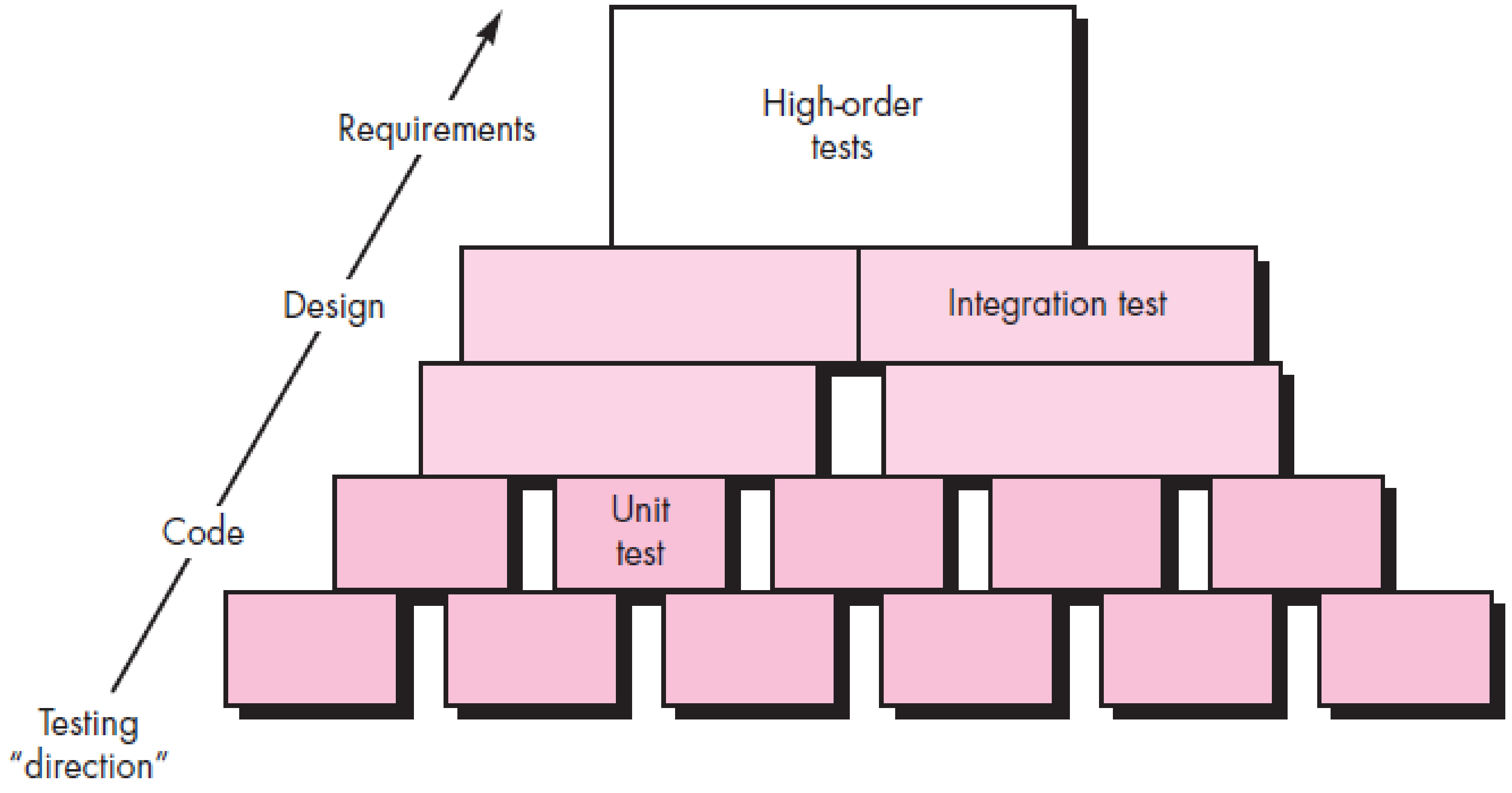
Testing – The Big Picture

- Unit testing begins at the vortex of the spiral and concentrates on each unit (e.g., component or class) of the software as implemented in source code.
- Testing progresses by moving outward along the spiral to integration testing, where the focus is on design and the architecture.

Testing – The Big Picture

- Taking another turn outward on the spiral, you encounter validation testing, where requirements established as part of requirements modeling are validated against the software that has been constructed.
- Finally, you arrive at system testing, where the software and other system elements are tested as a whole.

Software Testing Steps



Software Testing Steps

- Unit testing makes heavy use of testing techniques that exercise specific paths in a component's control structure to ensure complete coverage and maximum error detection.
- Next, components must be assembled or integrated to form the complete software package.
- Integration testing addresses the issues associated program construction.

Software Testing Steps

- After the software has been integrated, a set of high-order tests is conducted. Validation criteria (established during requirements analysis) must be evaluated.
- Validation testing provides final assurance that software meets all requirements.
- Software, once validated, must be combined with other system elements (e.g., hardware, people, databases).
- System testing verifies that all elements mesh properly and that overall system function/performance is achieved.

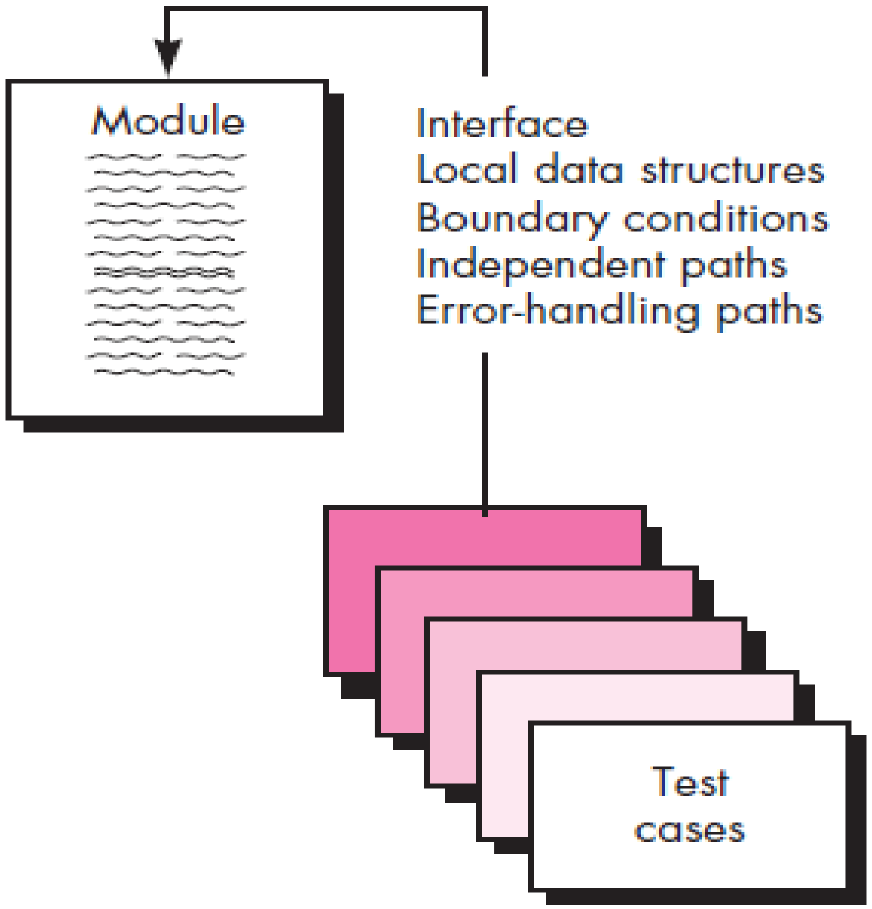
When Testing Should End?

- No definite answer
- Metrics can be used as base

- Testing entire system at the end
  - May produce too many errors
- Testing very frequently
  - Incurs overhead
- Correct Approach?
  - Test Units
  - Integrate Units and Test
  - Test Entire System

Unit Testing

- **Goal:** Test smallest Unit



Unit Test Considerations

- Interfaces:
  - To ensure that data properly flows in and out
- Local Data Structures:
  - To ensure integrity of stored data
- Independent Paths:
  - To ensure that all statements are properly executed
- Boundary Conditions:
  - To verify values at the boundary
- Error Handling Paths:
  - To check error handling part of the system

Unit Test Considerations

- Data flowing in and out must be tested first
- Local values affect global data structures, hence they must be tested
- All execution paths in code must be thoroughly tested
- Software often fails at the boundary (nth element of array, ith repetition of loop), hence must be tested
- To ensure clean termination for errors (called anti bugging), error paths need to be tested

When to design Unit Test cases?

- Unit test cases can be designed either
  - Before coding begins or
  - After coding is completed

- Unit Testing involves two major concepts:
  - Driver
  - Stub

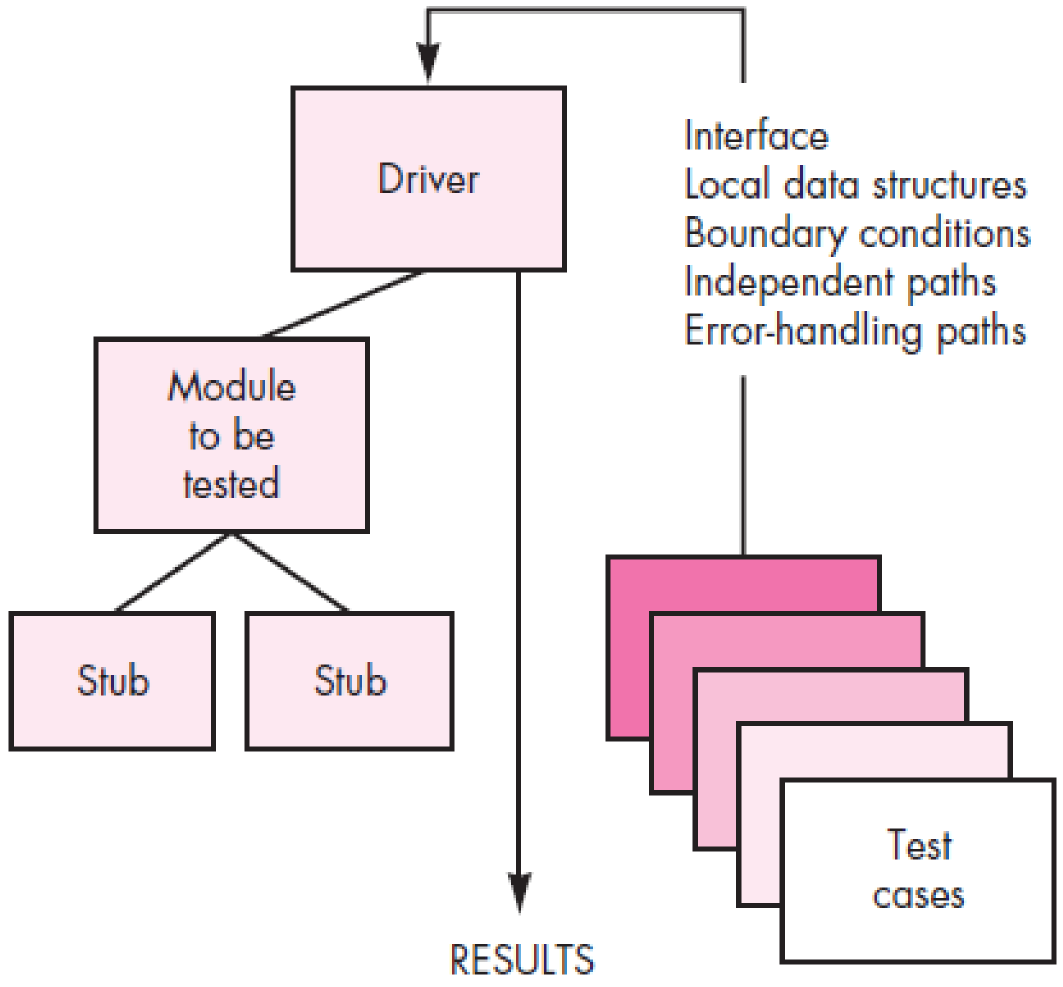
Drivers and Stubs

- "A component is not a stand alone program"
  - Cannot be tested by direct execution

- Driver is like a main program
  - Accepts test data
  - Passes the data to the module under test
  - Prints results

- Stubs replace the modules that are called by module under test
  - Dummy subprogram

Drivers and Stubs



Unit Testing: Summary

- Drivers and stubs represent testing “overhead.”
- That is, both are software that must be written but that is not delivered with the final software product.
- If drivers and stubs are kept simple, actual overhead is relatively low.
- Unit testing is simplified when a component with high cohesion is designed.
- When only one function is addressed by a component, the number of test cases is reduced and errors can be more easily predicted and uncovered.