Click here to review SW Testing Techniques

Software Testing Strategies

Chapter 18

Review SW Testing Techniques

Chapter 17

Software Testing Techniques

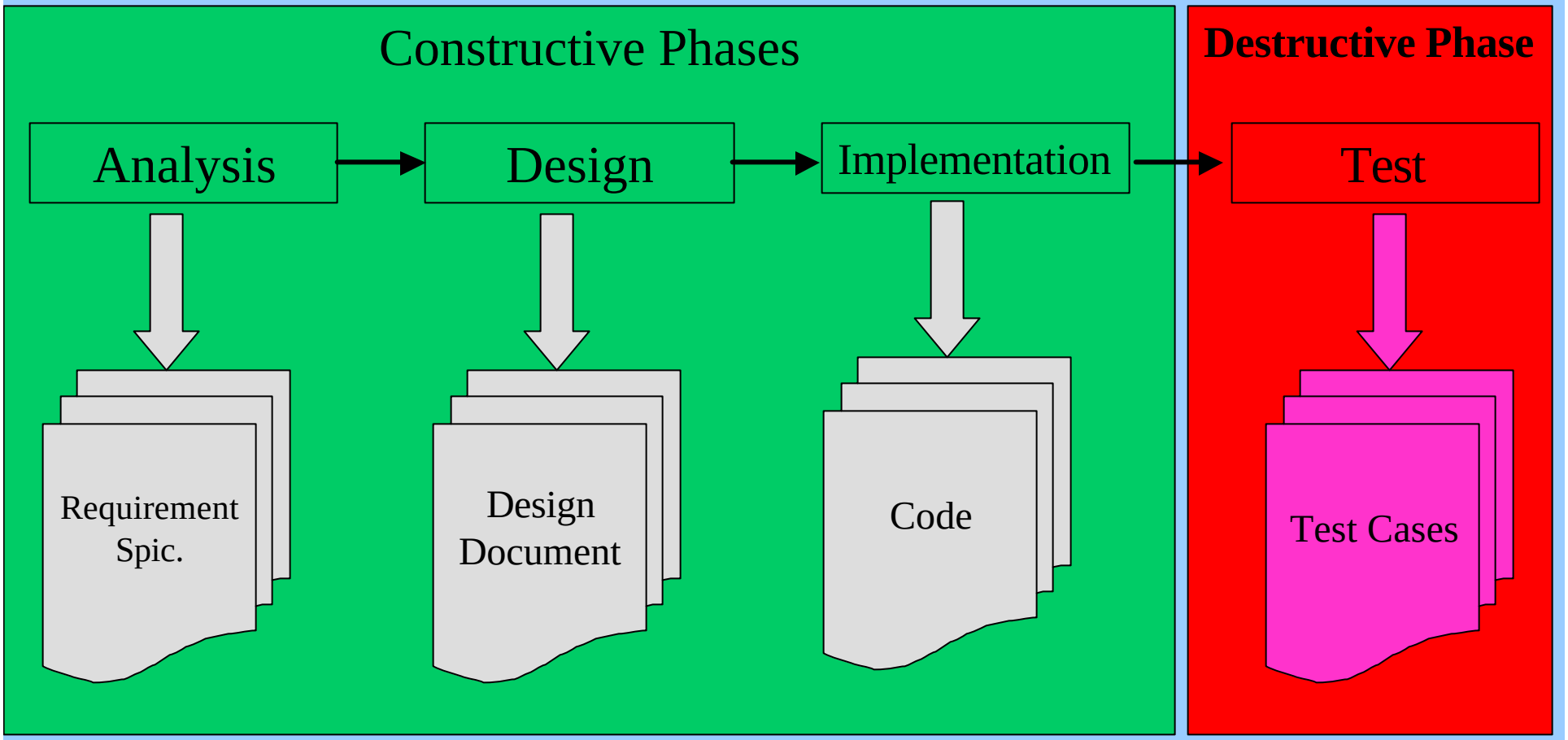
- Provide system guidance for designing tests that:
  - Exercise the internal logic of a program
    - “White Box” test cases design techniques
  - Exercise the input and output “Requirements” of a program
    - “Black Box”

To Uncover ERRORS / BUGS / MISUNDERSTANDING OF REQUIREMNTS ETC.

Software Testing Techniques

- Execute the program before the customer.
- To reduce the number of errors detected by customers.
- In order to find the highest possible number pf errors software testing techniques must be used

Software Testing Techniques



What is testing and why do we do it?

- Testing is the filter to catch defects before they are “discovered” by the customer
  - Every time the program is run by a customer, it generates a “test-case”.
  - We want our test cases to find the defects first.
- Software development is a human activity with huge potential for errors
- Testing before release helps assure quality and saves money

Test Cases Design

- Test cases should be designed to have the *highest* likelihood of finding problems
- Can test by either:
  - ***Black-box*** - using the specifications of what the software should do
    - Tests are derived from the I/O specification.
    - Used in most functional tests.
    - Other names: data-driven, input/output-driven.
  - ***White-Box*** - testing internal paths and working of the software
    - Examine internal program structure and derive tests from an examination of the program's logic.
    - Used to develop test cases for unit and integration testing
    - Other names: Glass-box, Logic-driven, Structural.

White-Box Testing

- Uses the control structure of the program/design to derive test cases
- We can derive test cases that:
  - Guarantee that all independent paths within a module have been visited at least once.
  - Exercise all logical decisions on their TRUE or FALSE sides
  - Execute all loops at their boundaries

A few White-Box Strategies

- Statement
  - Requires each statement of the program to be executed at least once.
- Branch
  - Requires each branch to be traversed at least once.
- Multi-condition
  - Requires each condition in a branch be evaluated.

More White-Box Strategies

- Basis Path
  - Execute all control flow paths through the code. Based on Graph Theory.
    - Thomas McCabe's Cyclomatic Complexity:
    - V(g) : #edges - #nodes + 2
    - Cyclomatic complexity is a SW metric that measures the complexity of a program.
    - The larger V(g) the more complex.
- Data Flow
  - Selects test data based on the locations of definition and the use of variables.

Statement Coverage

- The criterion is to require every statement in the program to be executed at least once
- Weakest of the white-box tests.
- Specified by the F.D.A. as the minimum level of testing.

Branch Coverage

- This criterion requires enough test cases such that each decision has a TRUE and FALSE outcome at least once.
- Another name: Decision coverage
- More comprehensive than statement coverage.

Branch Coverage

- Example:

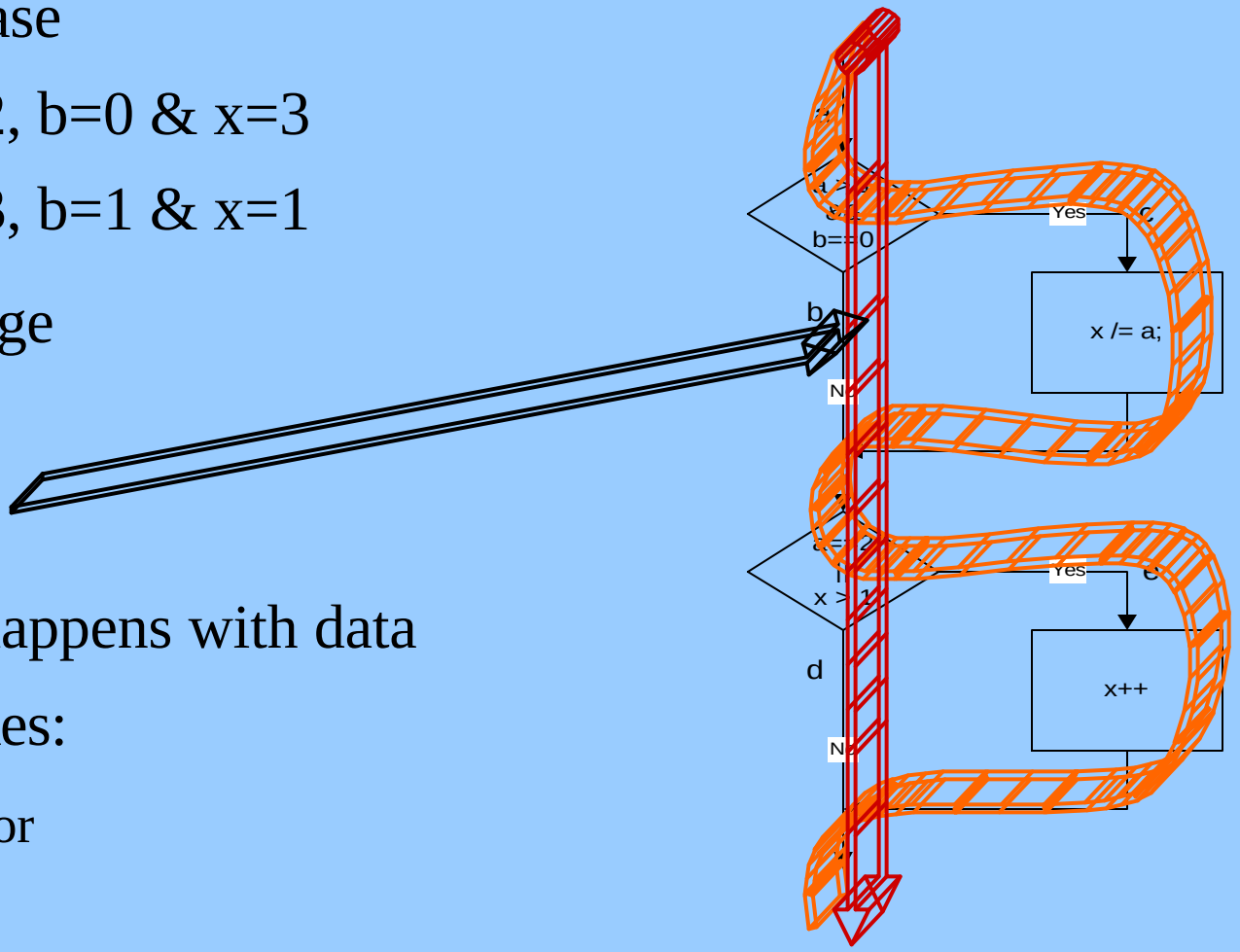
```
   void example(int a, int b, float *x)
   {
1    if ((a>1) && (b==0))
2       x /= a;
3    if ((a==2) || (x > 1)
4       x++;
   }
```

- Test case(s)

1. a=2, b=0, x=3
2. a=3, b=1, x=1

Branch Coverage

- Test Case
  1. a=2, b=0 & x=3
  2. a=3, b=1 & x=1
- Coverage
  1. ace
  2. abd
- What happens with data that takes:
  - –abe, or
  - –acd

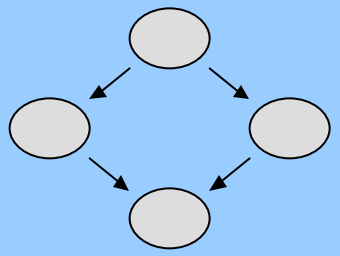


Basis Path

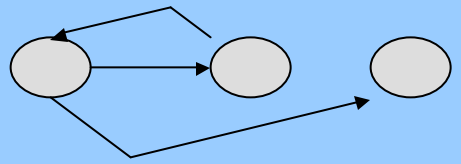
- Execute all *independent* flow paths through the code. Based on a flow graph.
  - An *independent* flow path is one that introduces at least 1 new set of statements or conditions
  - Must move along at least 1 new edge on *flow graph*
  - ***Flow graph*** shows the logical control flow using following notation:

Sequence

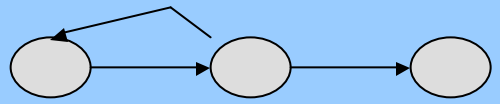
If



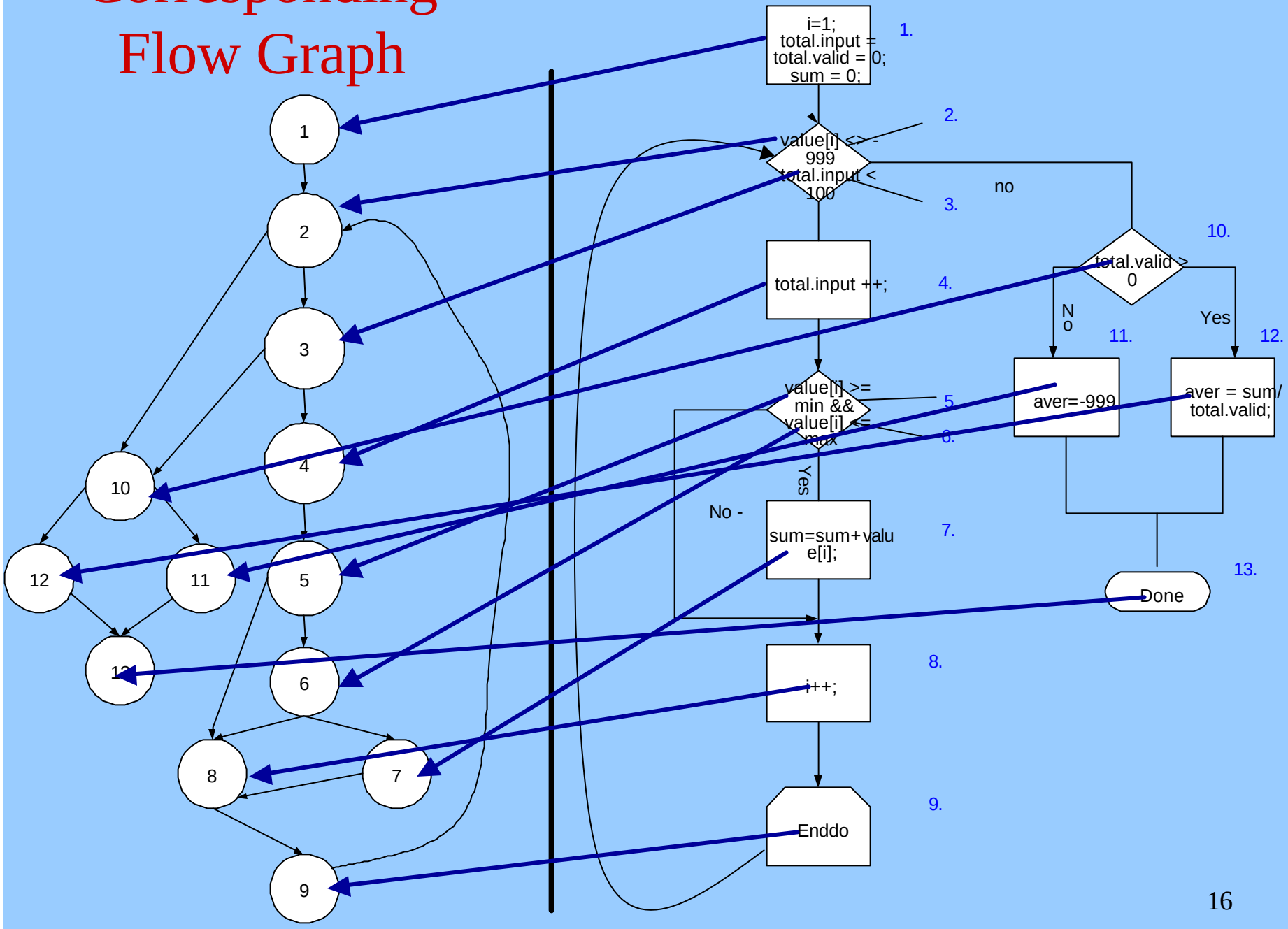
while



until



Corresponding Flow Graph

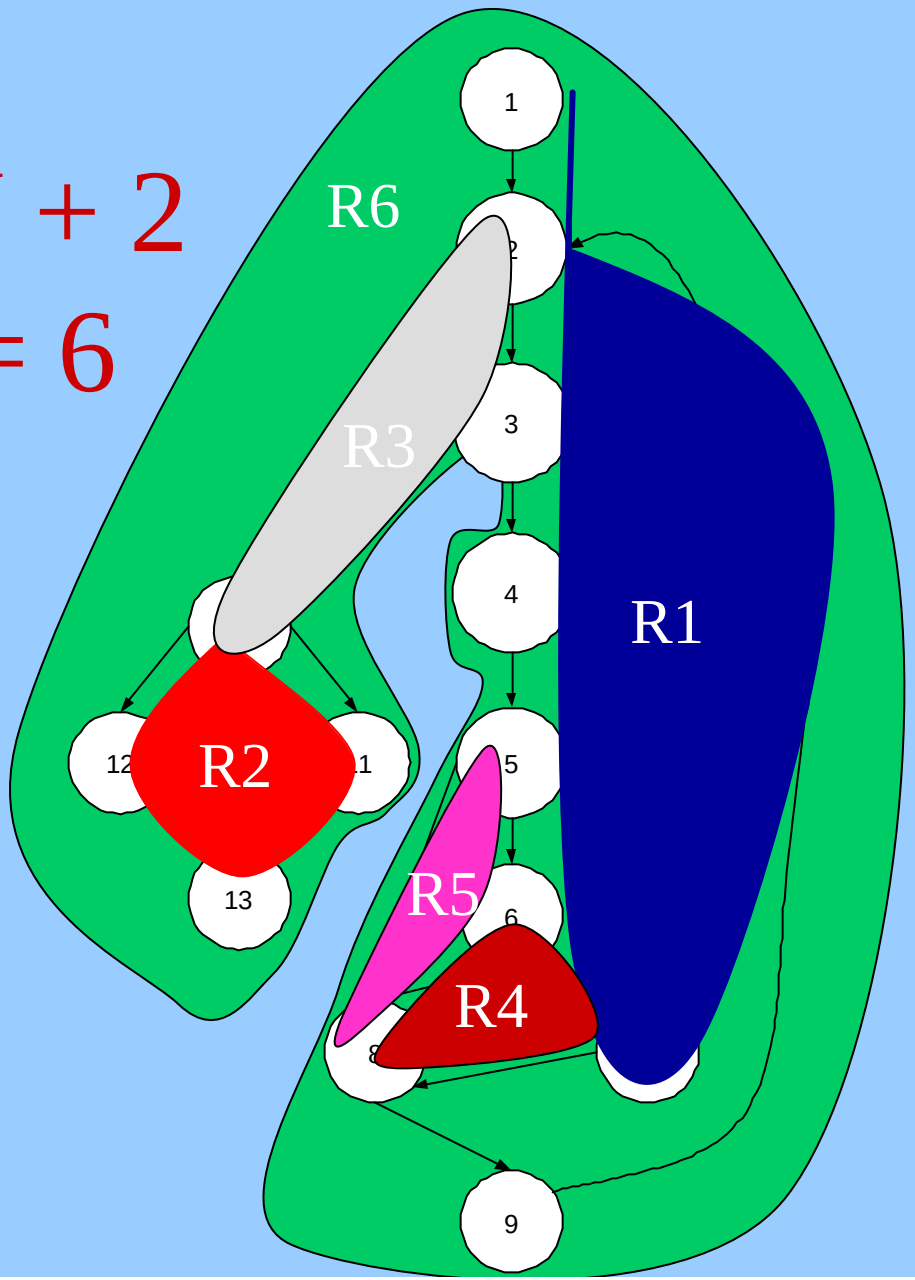


Number of Paths

$$V(g) = E - N + 2$$

$$17-13 + 2 = 6$$

$$R = 6$$



Black-Box Testing

- Focuses on functional requirements of the software without regard to the internal structure.
- data-driven, input/output-driven or behavior testing
- Used in most system level testing
  - Functional,
  - Performance
  - Recovery
  - Security & stress
- Tests set up to exercise full functional requirements of system

Black Box Testing Find Errors in ...

- Incorrect or missing functions (compare to white box)
- Interface errors
- Errors in External Data structures
- Behavior performance problems (Combinations of input make it behave poorly).
- Initialization and Termination errors (Sensitive to certain inputs (e.g., performance)
- Blackbox done much later in process than white box.

A few Black-box Strategies

- **Exhaustive input testing**
  - A test strategy that uses every possible input condition as a test case.
  - Ideal
  - Not possible!
- **Random**
  - Test cases are created from a pseudo random generator.
  - Broad spectrum. Not focused.
  - Hard to determine the result of the test.

Black-box Strategies

- **Equivalence Partitioning**
  - A black-box testing method that divides the input domain of a program into classes of data which test cases can be derived.
- **Boundary Value Analysis**
  - A test case design technique that complements equivalence partitioning, by selecting test cases at the “edges” of the class.

Boundary Value Analysis

- Experience shows that test cases *exploring boundary conditions* have a high payoff.
  - E.g., Most program errors occur in loop control.
- Different from equivalence partitioning:
  - Rather than *any* element in class, BVA selects tests at *edge* of the class.
  - In addition to input condition, test cases can be derived for output conditions.

- Similar to Equivalence partitioning. First identify Equivalence classes, then look at the boundaries.

Test Case Documentation

- Minimum information for a test case
  - Identifier
  - Input data
  - Expected output data
- Recommended to add the condition being tested (hypothesis).
- Format of test case document changes depending on what is being tested.
- Always include design worksheets.

Simple Test Case Format

Id	Condition	Input Data	Expected

Test Case Formats

- Testing worksheet
  - Test Case
    - Identifier (serial number)
    - Condition (narrative or predicate)
    - Input (Stimuli data or action)
    - Expected Output (Results)
  - Test Results
    - Actual Output (Results)
    - Status (Pass/Fail)

Use this Test Case format for your Project

Test Name/Number	
Test Objective	
Test Description	
Test Conditions	
Expected Results	
Actual Results	

ANSI/IEEE Test Case Outline

- Test-case-specification Identifier
  - A unique identifier
- Test Items
  - Identify and briefly describe the items and features to be exercised by this case
- Input Specifications
  - Specify each input required to execute the test case.
- Output Specifications
  - Specify all of the outputs and features required of the test items.

ANSI/IEEE Test Case Outline

- Environmental needs
  - Hardware
  - Software
  - Other
- Special procedural requirements
  - Describe any special constraints on the test procedures which execute this test case.
- Interfaces dependencies
  - List the id’s of test cases which must be executed prior to this test case

Software Testing Strategies
Chapter 18

Software Testing Strategies

- A formal plan for your tests.
  - What to test ?
  - What to test when new components are added to the system?
  - When to start testing with customer?
- Testing is a set of activities that can be planned in advance and conducted systematically.

Software Testing Strategies

- Testing begins “in the small” and progresses “to the large”.
- Start with a single component and move upward until you test the whole system.
- Early tests detects design and implementation errors, as move upward you start uncover errors in requirements .

Software Testing Strategies

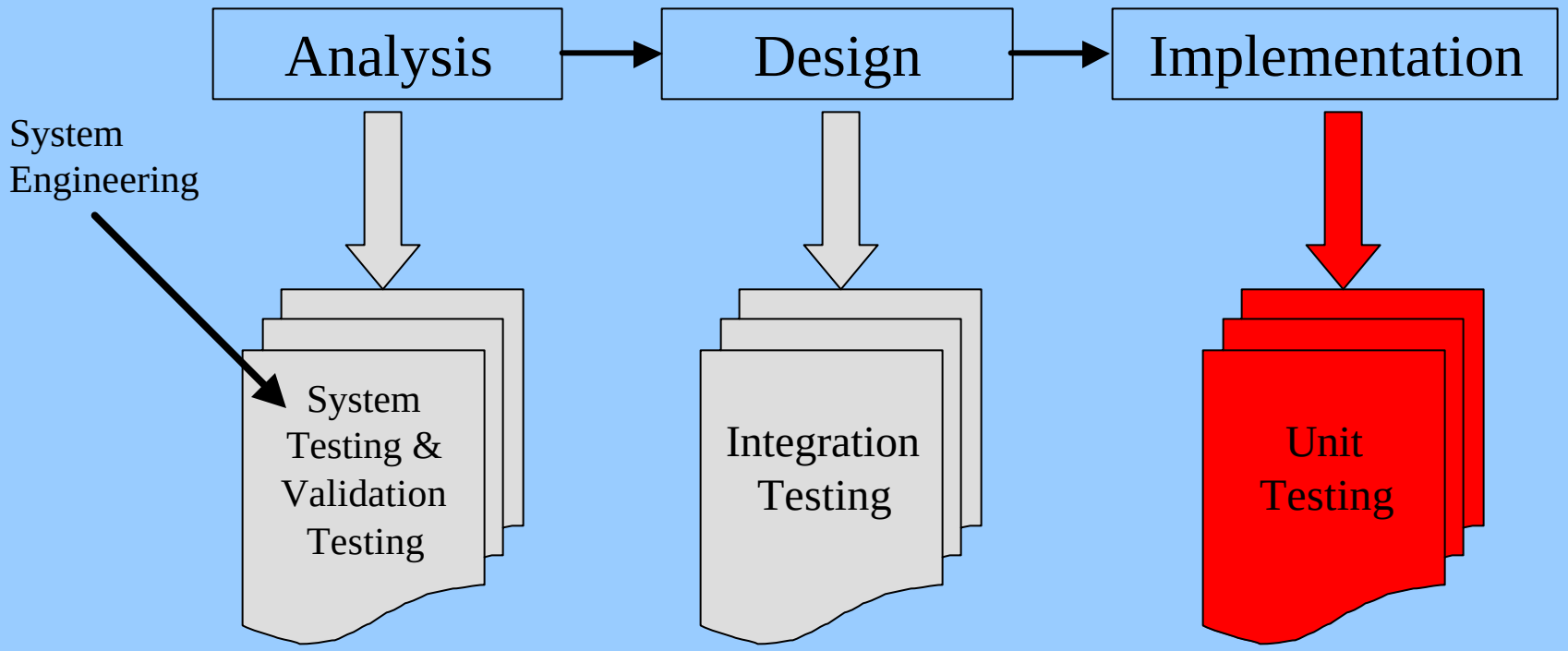
- Characteristics of testing strategies:
  - Testing begins at the component level, for OO at the class or object level, and works outward toward the integration of the entire system.
  - Different testing techniques, such as white-box and black-box, are appropriate at different times in the testing process.
  - For small projects, testing is conducted by the developers. For large projects, an independent testing group is recommended.
  - Testing and debugging are different activities, but debugging must be included in any testing strategy.

Testing can be used for verification and validation (V&V) of the Software:

- **Verification** - Are we building the product right?
  - Did the software correctly implements a specific function.
- **Validation** - Are we building the right product?
  - Has the software been built to customer requirements “Trace-ability”

The goal is to make sure that the software meets the organization quality measures

Conventional SW Testing can be viewed as a series of four steps:



Unit Testing

- Testing focuses on each component “unit” individually.
- Unit testing heavily depends on white-box testing techniques.
  - Tests of all components can be conducted in parallel.
  - Use the component level design description, found in the design document, as a guide to testing.

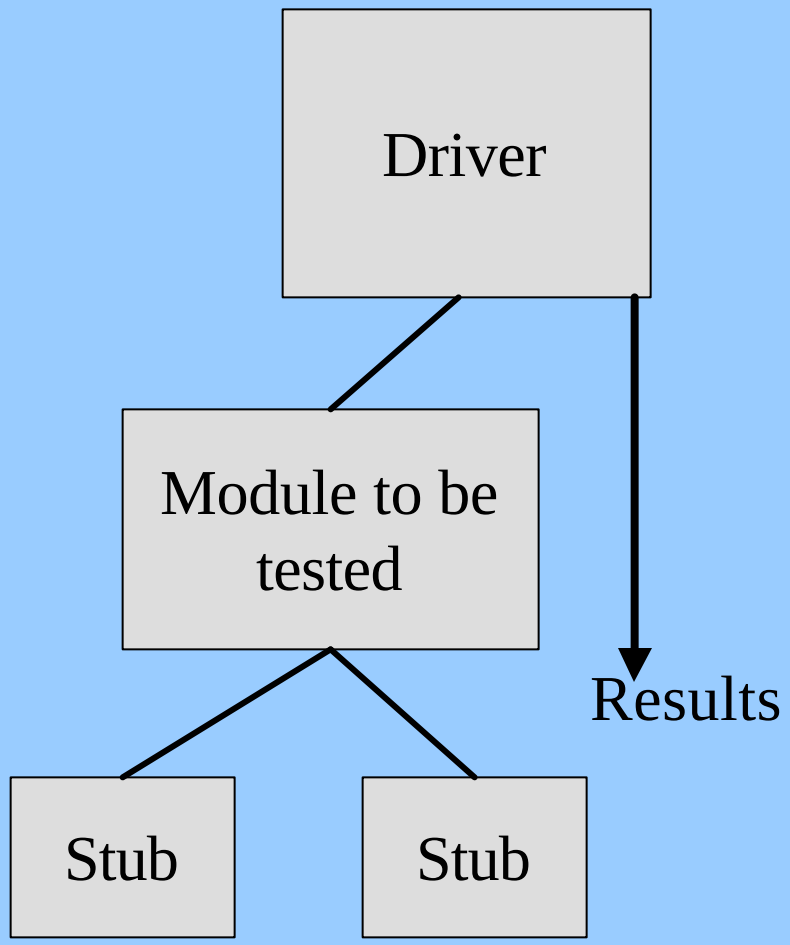
Unit Testing

- Things to consider as you write test cases for each component:
  - **Interface to the module** - does information flow in and out properly?
  - **Local data structures** - do local structures maintain data correctly during the algorithms execution?
  - **Boundary conditions** - all boundary conditions should be tested. Look at loops, First and last record processed, limits of arrays… This is where you will find most of your errors.
  - **Independent paths**- Try to execute each statement at least once. Look at all the paths through the module.
  - **Error Handling paths**- Trigger all possible errors and see that they are handled properly,
    - messages are helpful and correct,
    - exception-condition processing is correct and
    - error messages identify the location of the true error.

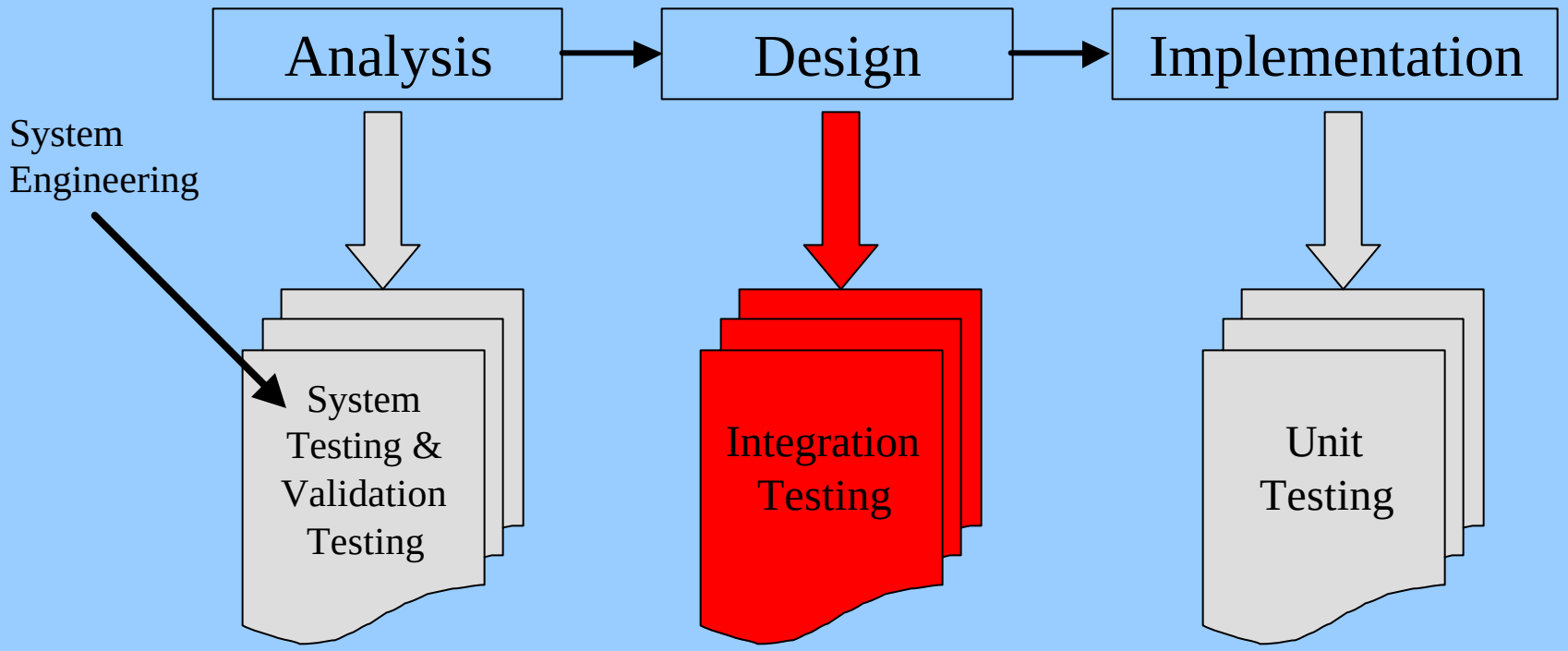
Unit Testing

- Things to consider as you write test cases for each component:
  - **Stubs and Drivers:**
    - Additional SW used to help test components.
  - **Driver –**
    - a main program that passes data to the component and displays or prints the results.
  - **Stub** -
    - a "dummy" sub-component or sub-program used to replace components/programs that are subordinate to the unit being tested.
    - The stub may do minimal data manipulation, print or display something when it is called and then return control to the unit being tested.

Stubs and Drivers



Conventional SW Testing can be viewed as a series of four steps:



Integration Testing

- Testing focuses on the design and the construction of the SW architecture.
  - This is when we fit the units together.
- Integration testing includes both
  - verification and
  - program construction.
- Black-box testing techniques are mostly used.

  Some white-box testing may be used for major control paths.
  - Look to the SW design specification for information to build test cases.

Integration Testing

- Different approaches to Integration:
  - Top-down
  - Bottom-up
  - Sandwich
  - Big Bang!

Top-down Approach

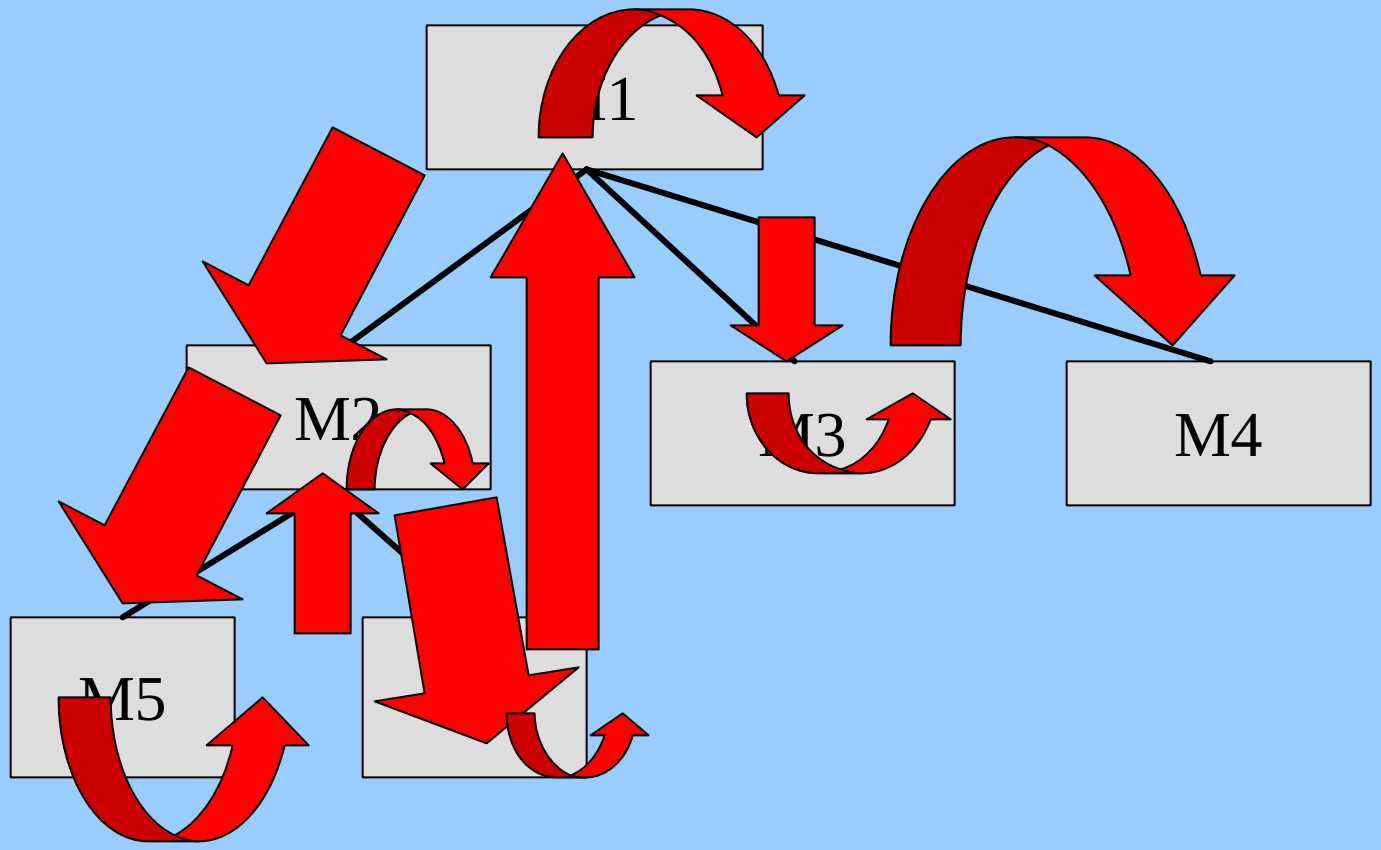
- Modules are integrated by moving down through the control hierarchy, beginning with the main program.

- Two approaches to Top-down integration:
  - Depth-first integration –
    - Integrate all components on a major control path.
  - Breadth-first integration –
    - Integrates all components at a level and then moves down.

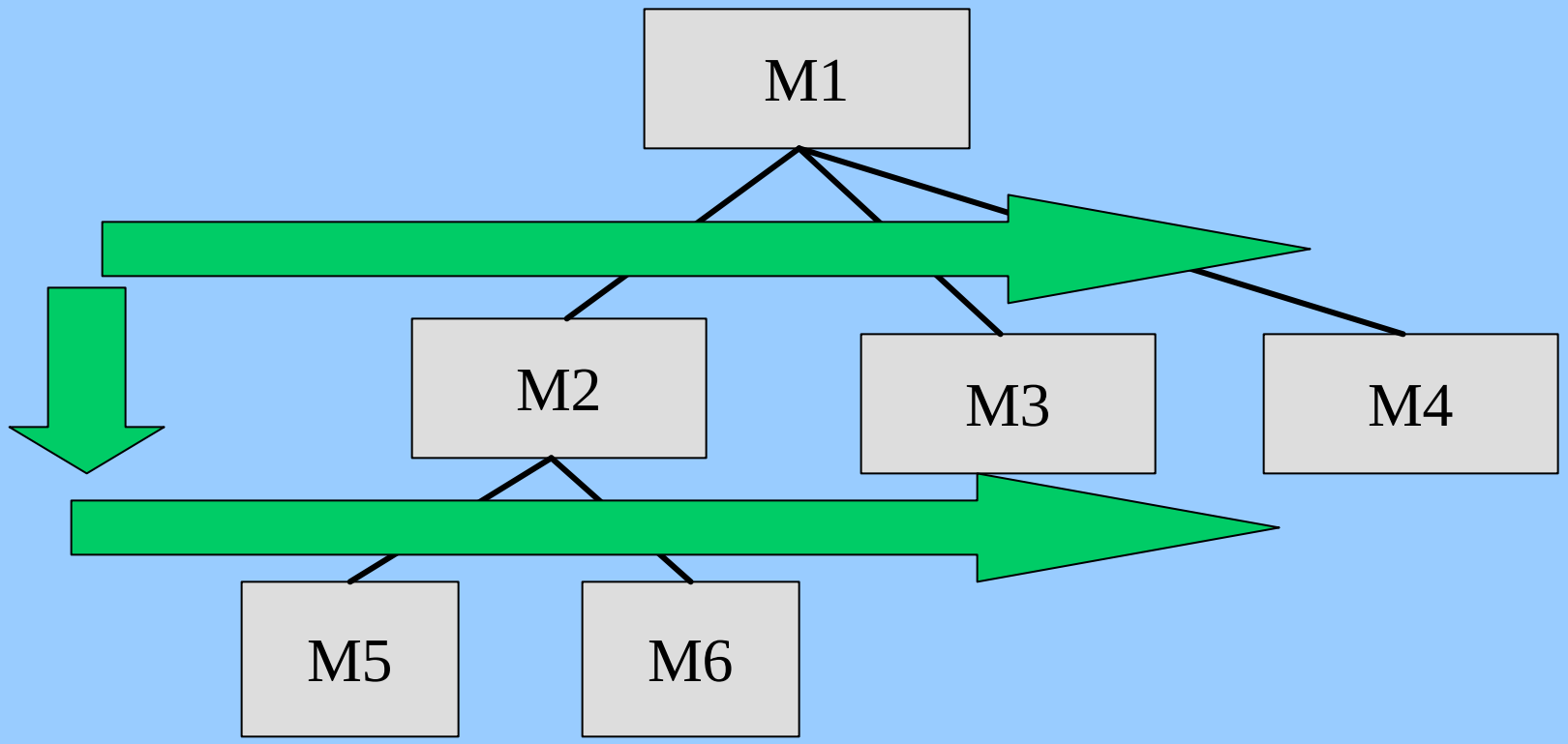
Top-down Integration Steps

- Main is used as a test driver and stubs are used for all other components
- Replace sub-ordinate stubs one at a time (the order depends on approach used depth or breadth) with actual components.
- retest as each component is integrated.
- On completion of a set of tests replace stubs
- To regret ion testing to make sure new modules didn't introduce new errors

Depth-first Example



Breadth-first integration



Stub As You Go

Top-down Integration

- Advantages:
  - Major control is tested first, since decision making is usually found in upper levels of the hierarchy.
  - If depth-first is used, a complete function of the SW will be available for demonstration.
- Disadvantages:
  - Many stubs may be needed.
  - Testing may be limited because stubs are simple. They do not perform functionality of true modules.

Integration Testing

- Different approaches to Integration:
  - Top-down
  - Bottom-up
  - Sandwich
  - Big Bang!

Bottom-up Approach

- Construction and testing begins at the lowest levels in the program structure.
- Implemented through the following steps:
  1. Low level components are combined into clusters or builds that perform a specific SW function.
  2. A Driver is written to control the test case input and output.
  3. The cluster or build is tested.
  4. Drivers are removed and the clusters or builds are combined moving upward in the program structure.

– GOTO TEXTBOOK PAGE 491

Bottom-up Approach

- Advantages:
  - Not need or for stubs.
- Disadvantage:
  - Controllers needed

Integration Testing

- Different approaches to Integration:
  - Top-down
  - Bottom-up
  - Sandwich
  - Big Bang!

Sandwich approach

- The sandwich approach is the best of both worlds, and minimizes the need for drivers and stubs.
- A top-down approach is used for the top components and bottom-up is used for the lower levels.

Big Bang approach

- Put the whole system together and watch the fire works!
  - (Approach used by may under-grads.)
  - Avoid this, and choose a strategy.

Regression Testing

- Each time the software changes I.e when new component is added.
- To make sure that new changes didn’t break existing functionality that used to work fine.
- Re-Execute some tests to capture errors as a result of side effects of new changes.

Validation Testing

Validation of the requirements

Validation Testing

- Testing ensures that all functional, behavioral and performance requirements of the system were satisfied.
  - These were detailed in the SW requirements specification.
  - Look in the validation criteria section.
- Only Black-box testing techniques are used.
- The goal is to prove conformity with the requirements.

Validation Testing

- For each test case, one of two possibilities conditions will exist:
  1. The test for the function or performance criteria will conform to the specification and is accepted.
  2. A deviation will exist and a deficiency is uncovered.
- This could be an error in the SW or a deviation from the specification
- SW works but it does not do what was expected.

Validation Testing

- Alpha and Beta Testing
  - A series of acceptance tests to enable the customer to validate all requirements.
  - Conducted by the end users and not the developer/tester/system engineer.
- Alpha
  - At the developer site by customer
- Bets
  - At the customer site by the end user.
  - Live testing

System Testing

Verifies that the new SW system integrates with the existing environment. This may include other systems, hardware, people and databases.

System Testing

- Black-box testing techniques are used.
- Put your software in action with the reset of the system.
- Many types of errors could be detected but who will accept responsibility?

System Testing Types

- **Recovery Testing** - force the SW to fail in a number of ways and verify that it recovers properly and in the appropriate amount of time.
- **Security Testing** - attempt to break the security of the system.
- **Stress Testing**- execute the system in a way that requires an abnormal demand on the quantity, frequency or volume of resources.
- **Performance Testing**- For real-time and embedded systems, test the run-time performance of the system.

System Testing Types

- **Regression Testing again**
  - **re-executing a subset of all test cases that have already been conducted to make sure we have not introduced new defects.**

Click here for OO Testing

Chapter 23