15-413

Lecture Notes on System Testing

Bernd Bruegge

School of Computer Science

Carnegie Mellon University

Pittsburgh PA 15213

17 November 1998

Odds and Ends

- Object Design Review
- Internal Project Review
  - **Status of each Subsystem**
  - **Demonstration (Discussion) of demos demonstrating work products from project agreement**
- Client Acceptance Test
  - **Preliminary Schedule and Speaker Assignments**
- Testing Manual Template

Object Design and Implementation Review

- November 24
- Purpose
  - **Sanity Check with Project Agreement**
  - **Review of Analysis, System Design**
  - **Presentation of Object Design**
  - **Publication of APIs**

Outline of Object Design Review Presentations

- Scenarios from Project Agreement
- Subsystem Object Model (RAD)
- Hardware/Software Allocation (SDD)
- Algorithm and Data Structure Decisions (ODD)
- Code Example (Stub) (Implementation)
- Services provided by Subsystem
- Services needed by Subsystem
- Status of Subsystem Development

Test Manual Document

- **Template out today**
- **Unit Test Manual:**
  - **Each team describes and submits a small set of representative test cases for their subsystem unit test (at least 2 test cases)**
- **System Test Manual:**
  - **Each team is involved in the description and implementation of a double or a triple subsystem test**

The End of the Tunnel (slightly curved)

- 11/17 System testing (today)
- 11/19 Middleware
- 11/24 Object design review (Speakers needed)
- 12/1 Software lifecycle revisited
- 12/3 Guest lecture: Machine learning
- 12/8 Client acceptance test dry run
- 12/10 Client Acceptance Test

Upcoming Deadlines

- November 23, 3pm:
  - **Object Design Document (ODD)**
  - **Revised (if necessary) SDD**
- November 24
  - **Final Homework out**
- December 3, 6pm
  - **Unit Test Manual due**
- December 10, 4pm
  - **System Test Manual due**
- December 16, 6pm
  - **Final Homework due**

Client Acceptance Test : Presentations

- Where: Rangos Room, University Center
- (20 min) Requirements (1 speaker)
  - **Functional and global requirements, constraints**
  - **3 Demos**
- (20 min) System Design (1 speaker)
  - **System Design issues**
- Demo 1 (2 speakers)
- Demo 2 (2 speakers)
- Demo 3 (2 speakers)
- For each Demo
  - **Actual demonstration of selected scenario**
  - **Screen snapshots as backup**

Object Design and Implementation Review

- One speaker per team still needed

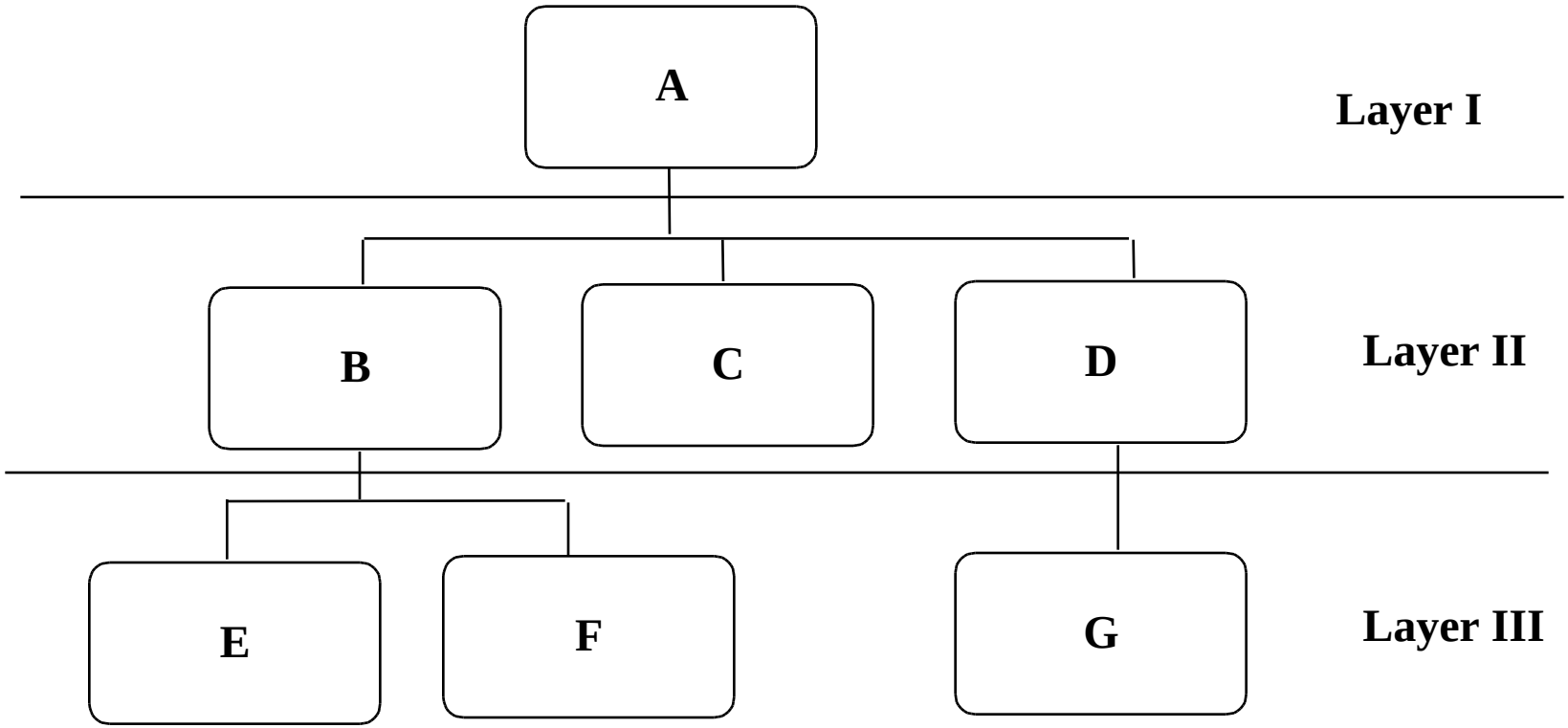
Testing Manual

- The test manual consists of a set of separate documents
  - **Must be in HTML format**
  - **Each team is responsible for their test cases**
  - **One document from each team (posted on document discuss)**
- Test manual template placed on 15-413 Home page by end of today (Test Manual in Work Products)

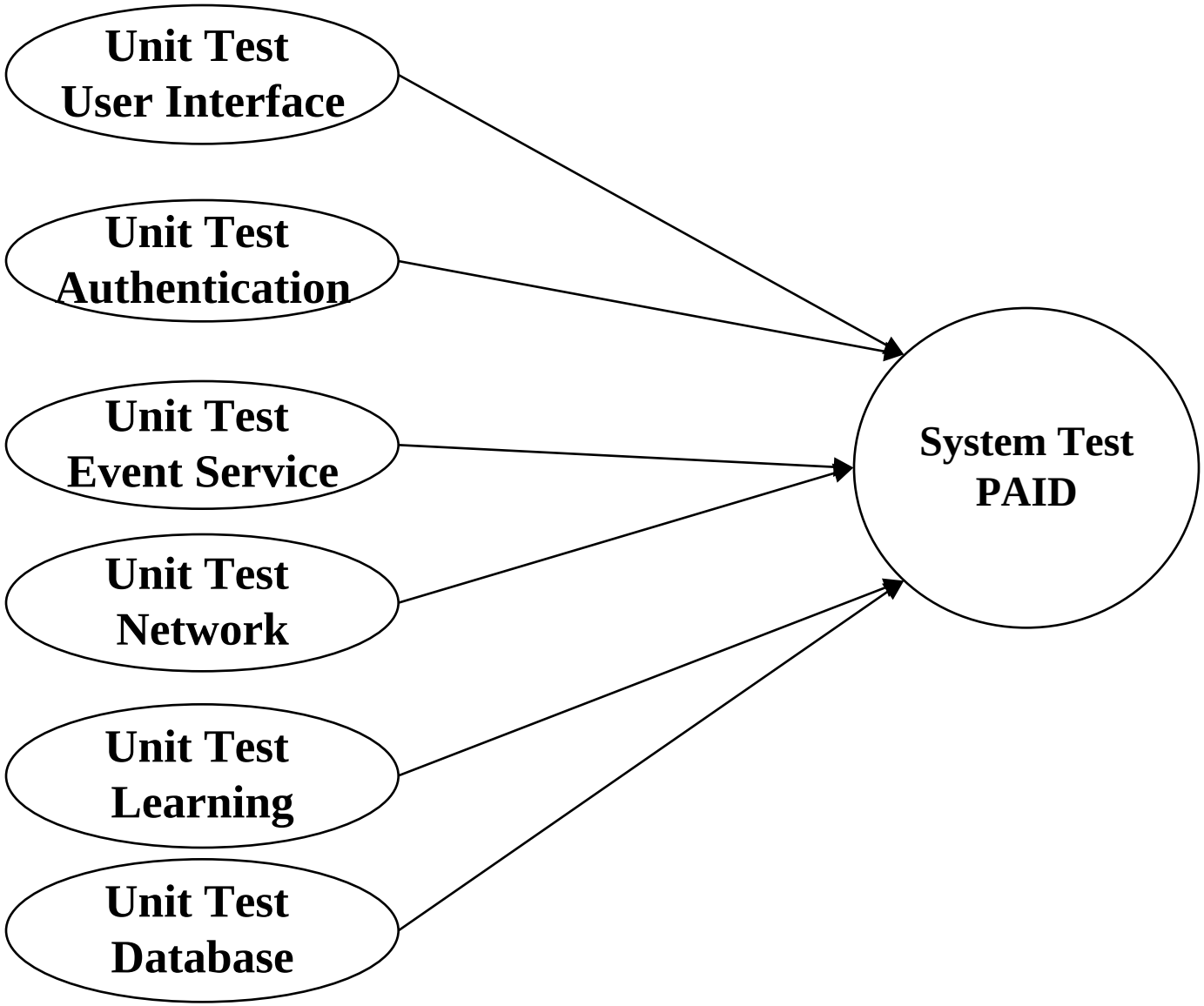
Testing: Integration Strategies

- The entire system is viewed as a collection of subsystems (sets of classes) determined during the system and object design.
- The order in which the subsystems are selected for testing and integration determines the testing strategy
  - **Big bang integration (Non-incremental)**
  - **Bottom up integration**
  - **Top down integration**
  - **Sandwich testing**
  - **Variations of the above**
- The selection is based on the system decomposition and the "Calls" Association (from the SDD)

Example: Call Hierarchy with 3 Layers



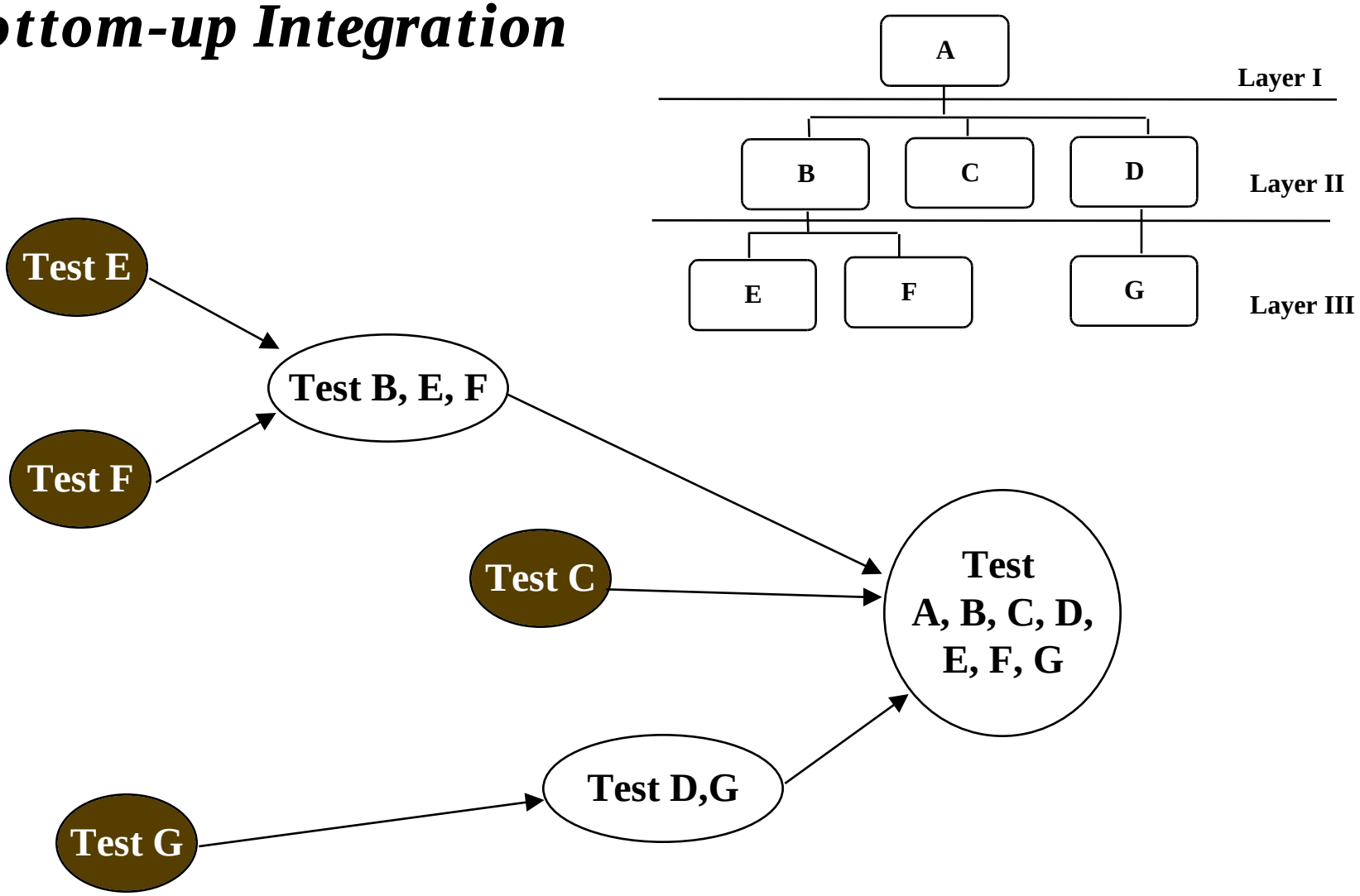
Integration Testing: Big-Bang Approach



Bottom-up Testing Strategy

- The subsystem in the lowest layer of the call hierarchy are tested individually
- Then the next subsystems are tested that call the previously tested subsystems
- This is done repeatedly until all subsystems are included in the testing
- Special program needed to do the testing
  - ***Test Driver*: A routine that calls a particular subsystem and passes a test case to it**

Bottom-up Integration



Pros and Cons of bottom up integration testing

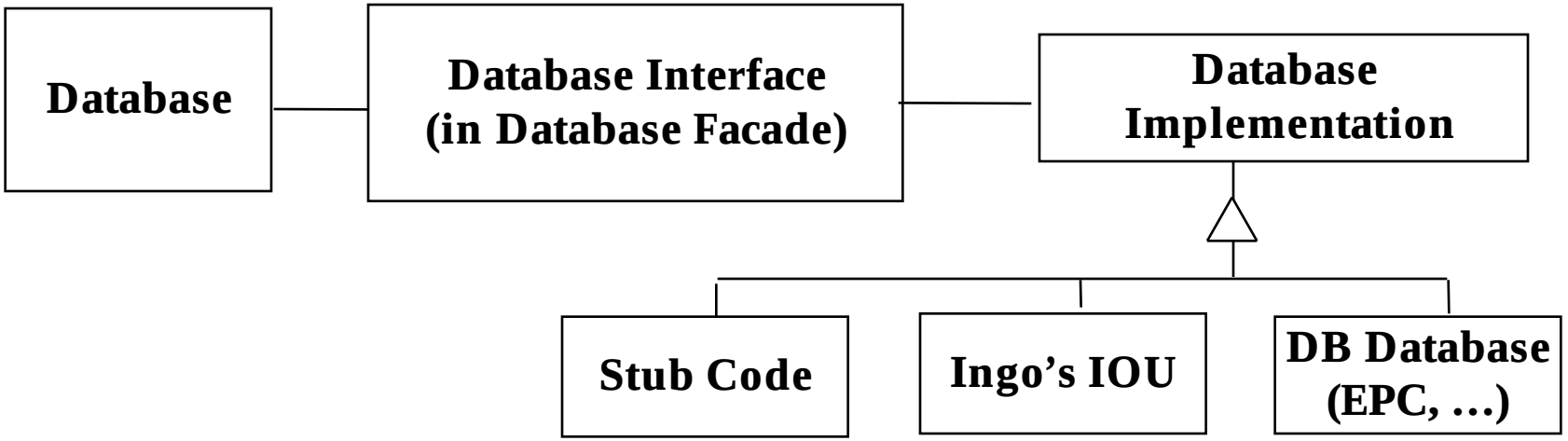
- Bad for functionally decomposed systems:
  - **Tests the most important subsystem last**
- Useful for integrating the following systems
  - **Object-oriented systems**
  - **Systems with strict performance requirements such as real-time systems**

Top-down Testing Strategy

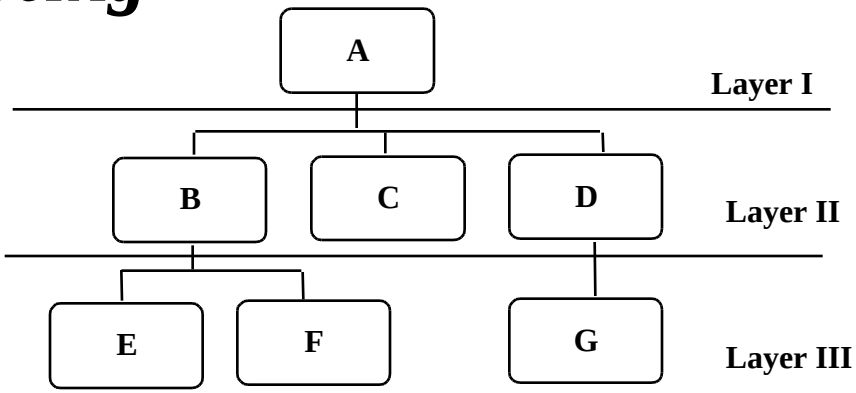
- Test the top layer or the controlling subsystem first
- Then combine all the subsystems that are called by the tested subsystems and test the resulting collection of subsystems
- Do this until all subsystems are incorporated into the test
- Special program needed to do the testing:
  - ***Test stub:*** **A program or a method that simulates the activity of a missing subsystem by answering to the calling sequence of the calling subsystem and returning back fake data.**

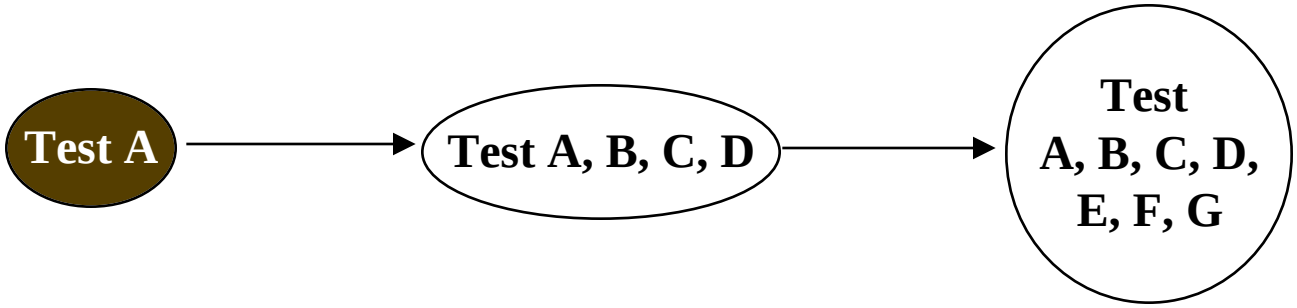
Using the Bridge Pattern for Top-Down Integration Testing

- Use the bridge pattern to provide multiple implementations under the same interface.
- Interface to a component that is incomplete, not yet known or unavailable during testing



Top-down Integration Testing





Layer I

Layer I + II

All Layers

Pros and Cons of top-down integration testing

- ☺ Test cases can be defined in terms of the functionality of the system (functional requirements)
- ☹ Writing stubs can be difficult: Stubs must allow all possible conditions to be tested.
- ☹ Possibly a very large number of stubs may be required, especially if the lowest level of the system contains many methods.
- One solution to avoid too many stubs: *Modified top-down testing strategy*
  - **Test each layer of the system decomposition individually before merging the layers**
  - **Disadvantage of modified top-down testing: Both, stubs and drivers are needed**

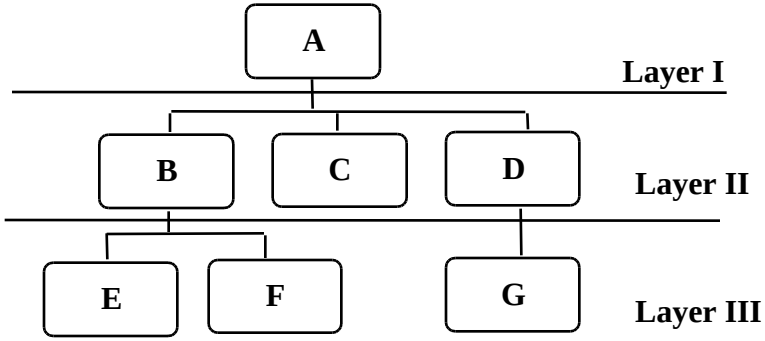
Sandwich Testing Strategy

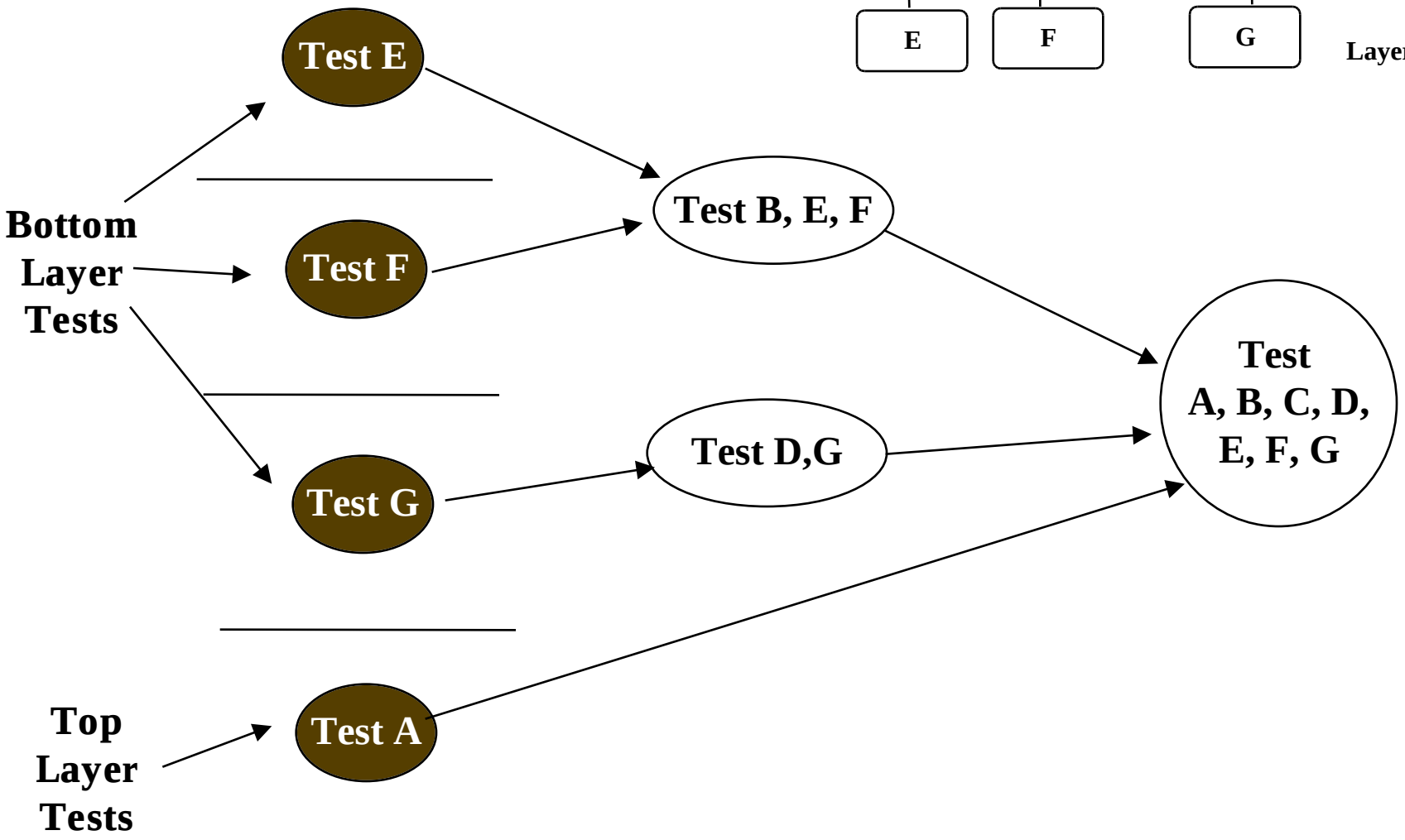
- Combines top-down strategy with bottom-up strategy
- *The system is viewed as having three layers*
  - **A target layer in the middle**
  - **A layer above the target**
  - **A layer below the target**
  - **Testing converges at the target layer**
- How do you select the target layer if there are more than 3 layers?
  - **Heuristic: Select a layer that minimizes the number of stubs and drivers**

Selecting Layers for the PAID system

- Top Layer:
  - **User Interface, Authentication, Learning**
- Middle Layer:
  - **Network, Event Service**
- Bottom Layer
  - **Database**

Sandwich Testing Strategy





Pros and Cons of Sandwich Testing

☺ Top and Bottom Layer Tests can be done in parallel

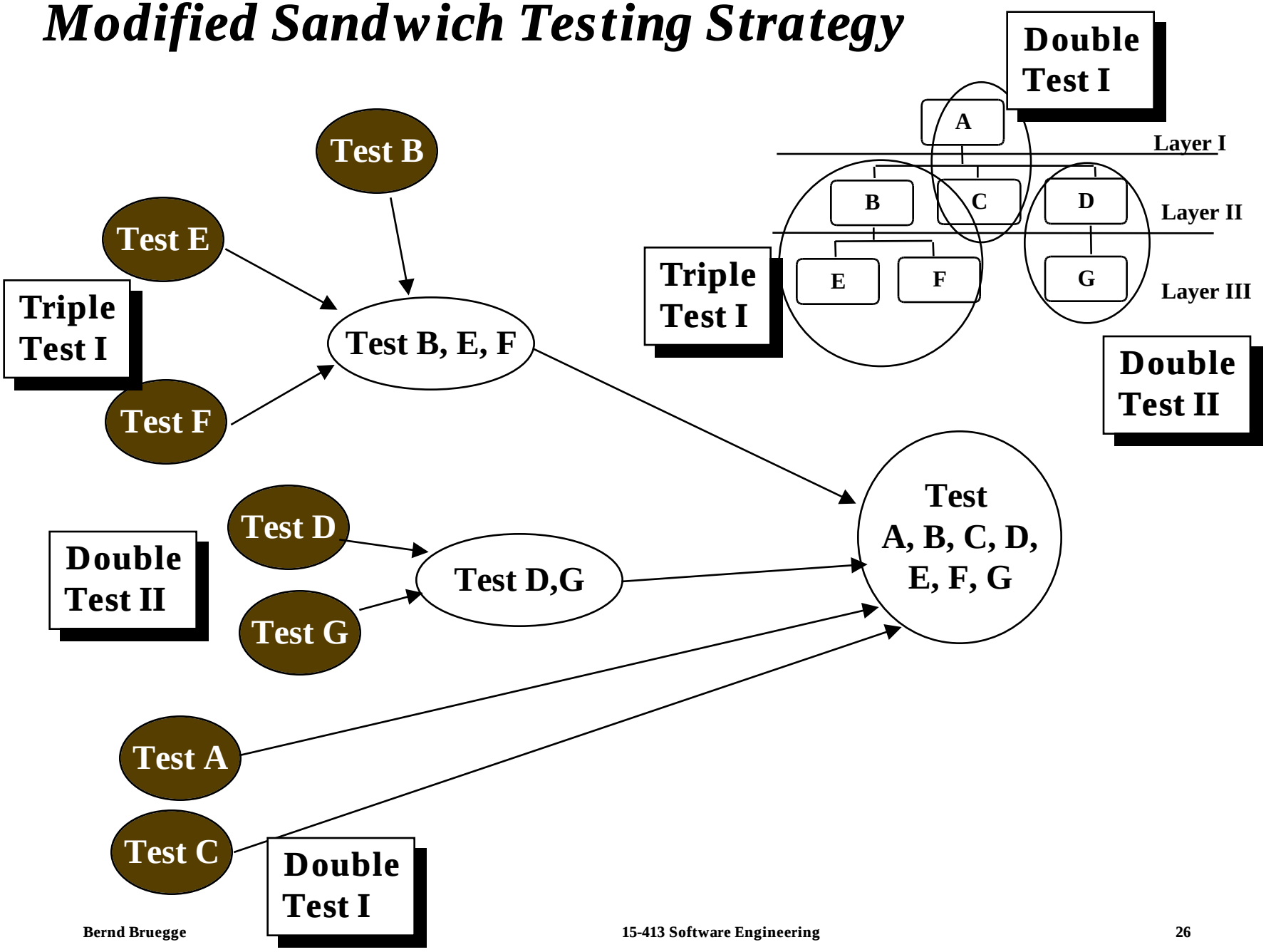
☹ Does not test the individual subsystems thoroughly before integration

❖ Solution: Modified sandwich testing strategy

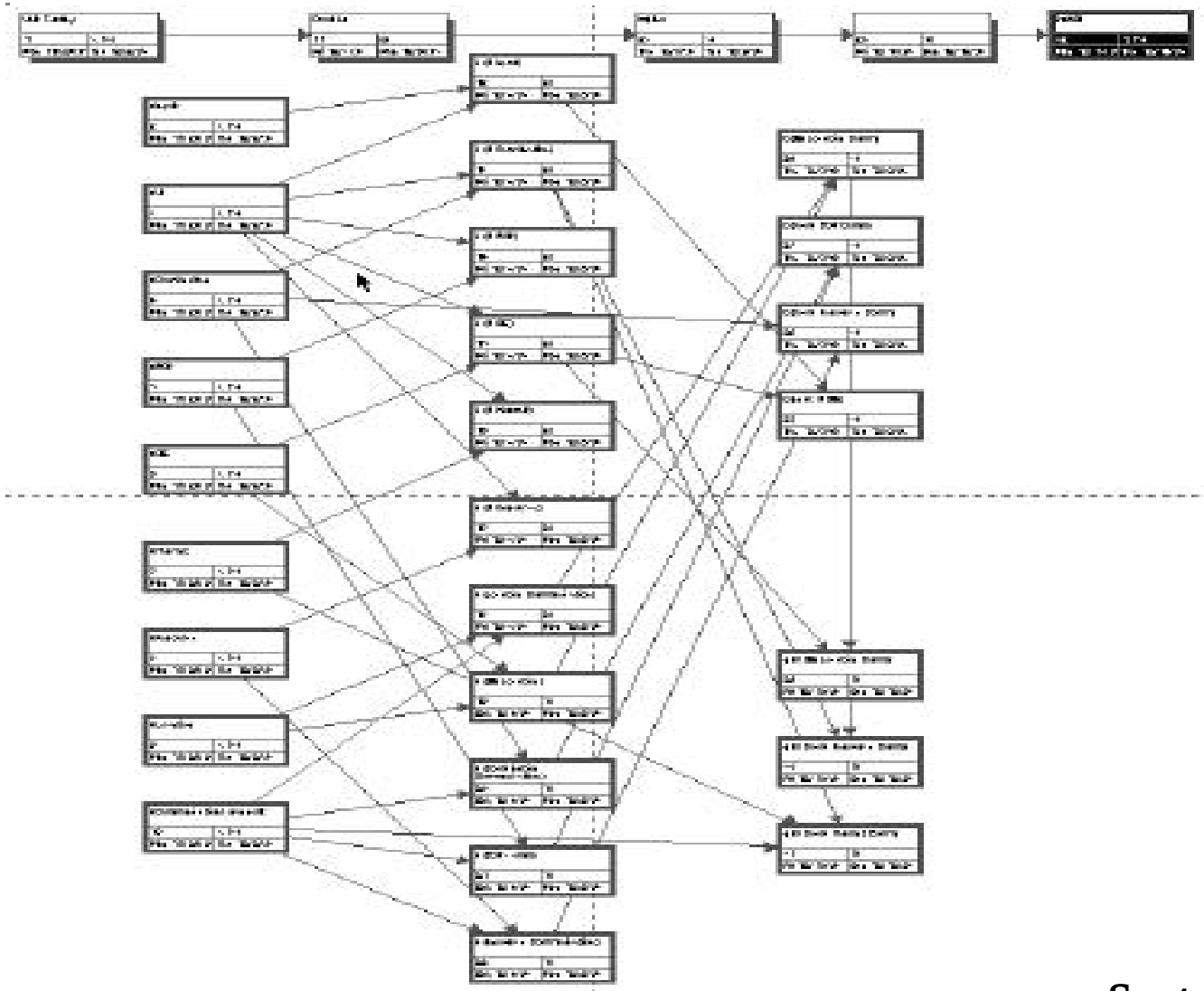
Modified Sandwich Testing Strategy

- Test in parallel:
  - **Middle layer with drivers and stubs**
  - **Top layer with stubs**
  - **Bottom layer with drivers**
- Test in parallel:
  - **Top layer accessing middle layer (top layer replaces drivers)**
  - **Bottom accessed by middle layer (bottom layer replaces stubs)**

Modified Sandwich Testing Strategy



Scheduling Sandwich Tests: 15-413 Fall 95



Unit Tests **Double Tests** **Triple Tests** **SystemTests**

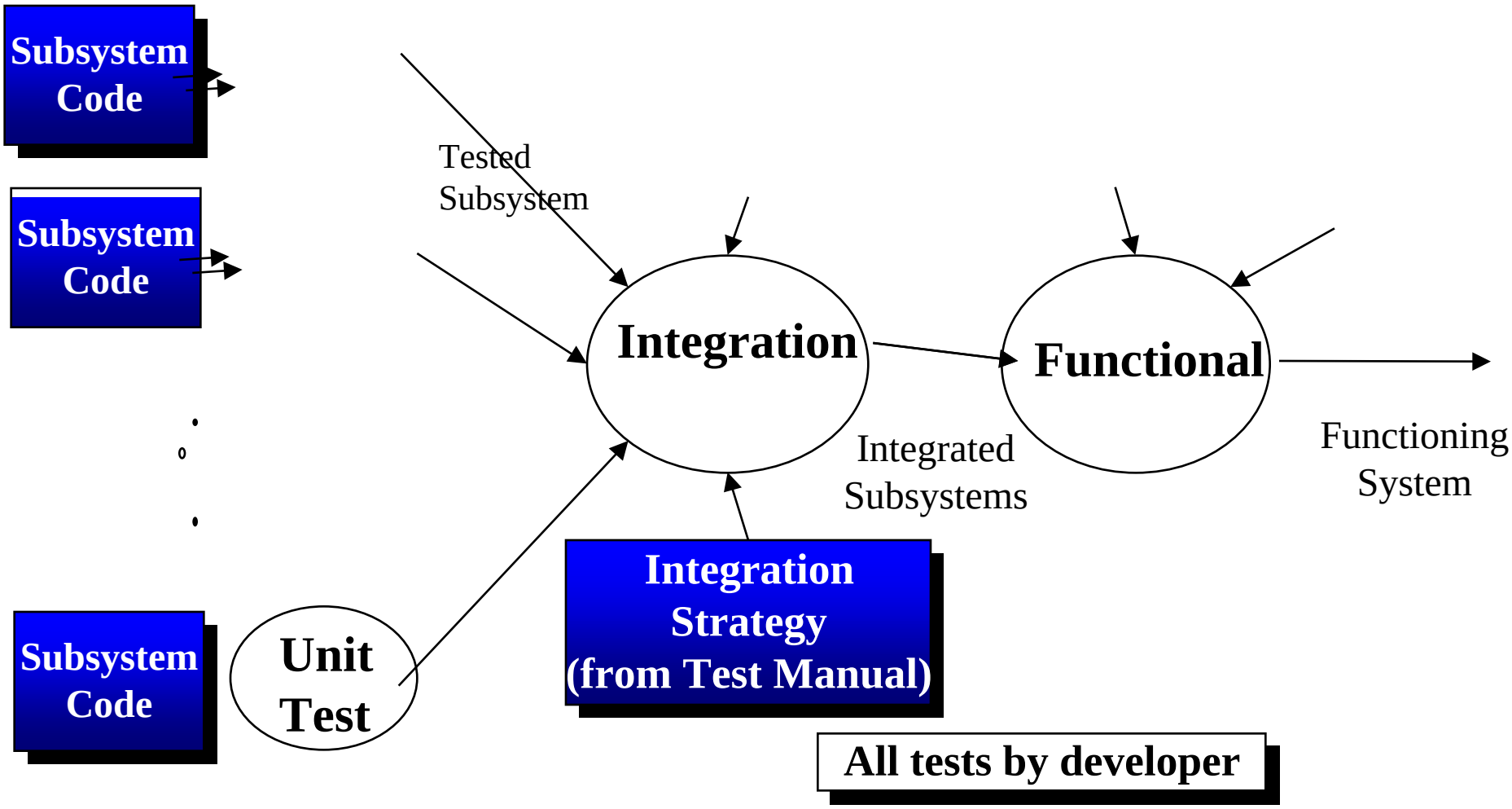
How do you choose an Integration Strategy?

- Factors to consider
  - **Amount of test harness (stubs &drivers)**
  - **Location of critical parts in the system**
  - **Availability of hardware**
  - **Availability of subsystems**
  - **Scheduling concerns**
- Bottom up approach
  - ☺**good for object oriented design methodologies**
  - ☹**Test driver interfaces must match module interfaces**
- Bottom up approach ctd
  - ☹**Top-level modules are usually important and cannot be neglected up to the end of testing**
  - ☹ **Detection of user interface design errors postponed until end of testing**
- Top down approach
  - ☺**Test cases can be defined in terms of functions examined**
  - ☹**Need to maintain correctness of test stubs**
  - ☹**Writing stubs can be difficult**

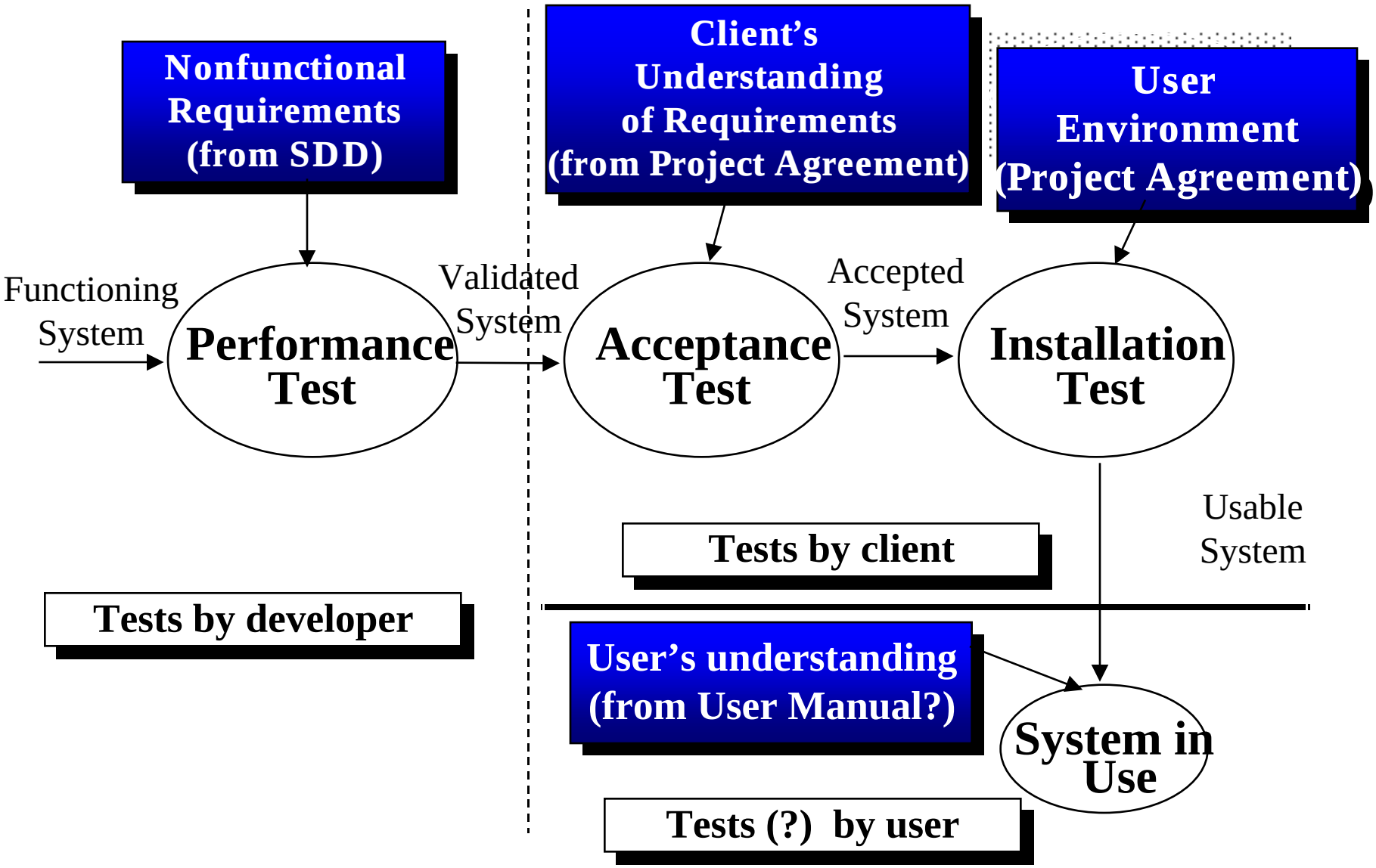
System Testing

- Structure Testing
- Functional Testing
- Performance Testing
- Acceptance Testing
- Installation Testing

System Testing Phases



System Testing Phases ctd



Structure Testing

Essentially the same as white box testing.

- Goal: Cover all paths in the system design
  - **Exercise all input and output parameters of each module.**
  - **Exercise all modules and all calls (each module is called at least once and every module is called by all possible callers.)**
  - **Use conditional and iteration testing as in unit testing.**
- Transaction flow diagram (structure test case)
  - **Transaction: Set of activities associated with a particular class of input.**
  - **Transaction flow diagram represents the sequence of steps or activities associated with the processing of the transaction.**
  - **Example of a transaction flow diagram: A list of modules called during the processing of the transaction : Table, Graph, etc.**

Functional Testing

Essentially the same as black box testing

- Goal: Test functionality of system
- Test cases are designed from the requirements analysis document (better: user manual) and centered around requirements and key functions (use cases)
- The system is treated as black box.
- Unit test cases can be reused, but in general new test cases have to be developed.

Performance Testing

Quality of requirements determines the ease of performance tests:

- **The more explicit the nonfunctional requirements, the easier they are to test.**

- Stress Testing
  - **Stresses limits of system (maximum number of users, peak demands, extended operation)**
- Volume testing
  - **Tests what happens if large amounts of data are handled**
- Configuration testing
  - **Tests the various software and hardware configurations**
- Compatibility test
  - **Tests backward compatibility with existing systems**

Performance Testing ctd

- Security testing
  - **Ensures security requirements are met**
- Timing testing
  - **Evaluates response times and time to perform a function**
- Environmental test
  - **Tests tolerances for heat, humidity, motion, portability**
- Quality testing
  - **Tests reliability, maintainability and availability of the system**
- Recovery testing
  - **Tests system's response to presense of errors or loss of data.**

Performance Testing ctd

- Documentation testing
  - **Insures the required documents are written and are consistent, accurate and easy to use**
  - **Functions described but not implemented**
  - **Functions implemented but not described**
  - **Inconsistencies between requirements and user manual**
- Human factors testing
  - **Tests user interface to system**

Test Cases for Performance Testing

- Push the (integrated) system to its limits.
- Goal: Try to break the subsystem
- Test how the system behaves when overloaded.
  - Can bottlenecks be identified? (First candidates for redesign in the next iteration
- Try unusual orders of execution
  - Call a receive() before send()
- Check the system's response to large volumes of data
  - If the system is supposed to handle 1000 items, try it with 1001 items.
- What is the amount of time spent in different use cases?
  - Are typical cases executed in a timely fashion?

Steps in Integration Testing

- 1. Based on the integration strategy, *select a subsystem* to be tested. Unit test all the classes in the subsystem.
- 2. Put selected subsystem together; do any *preliminary fix-up* necessary to make the integration test operational (drivers, stubs)
- 3. Do *functional testing:* Define test cases that exercise all uses cases with the selected subsystems
- 4. Do *structural testing:* Define test cases that exercise the selected subsystems
- 5. Execute *performance tests*
- 6. *Keep records* of the test cases and testing activities.
- 7. Based on the integration strategy, *integrate the next set* of subsystems and repeat steps 1 to 7.

The primary *goal of integration testing is to identify errors* in the (current) subsystem configuration.

Acceptance Testing

- Goal: Demonstrate system is ready for operational use
  - **Choice of tests is made by client/sponsor**
  - **Many tests can be taken from integration testing**
  - **Acceptance test is performed by the client, not by the developer.**
- Majority of all bugs in software is typically found by the client after the system is in use, not by the developers or testers. Therefore two kinds of additional tests:

- *Alpha test:*
  - **Sponsor uses the software at the *developer's site.***
  - **Software used in a controlled setting, with the developer always ready to fix bugs.**
- *Beta test:*
  - **Conducted at *sponsor's site* (developer is not present)**
  - **Software gets a realistic workout in target environment**
  - **Potential customer might get discouraged**

Test Life Cycle

Establish the test objectives

Design the test cases

Write the test cases

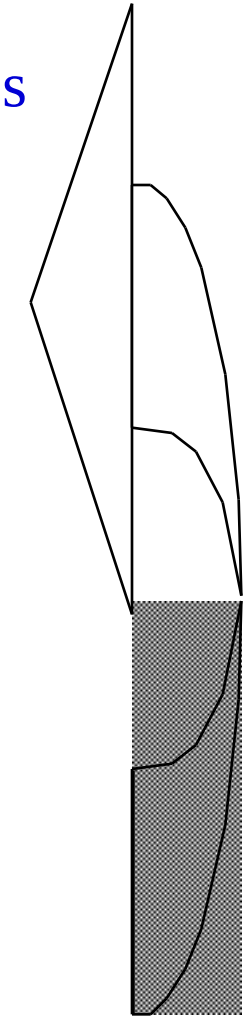
Test the test cases

Execute the tests

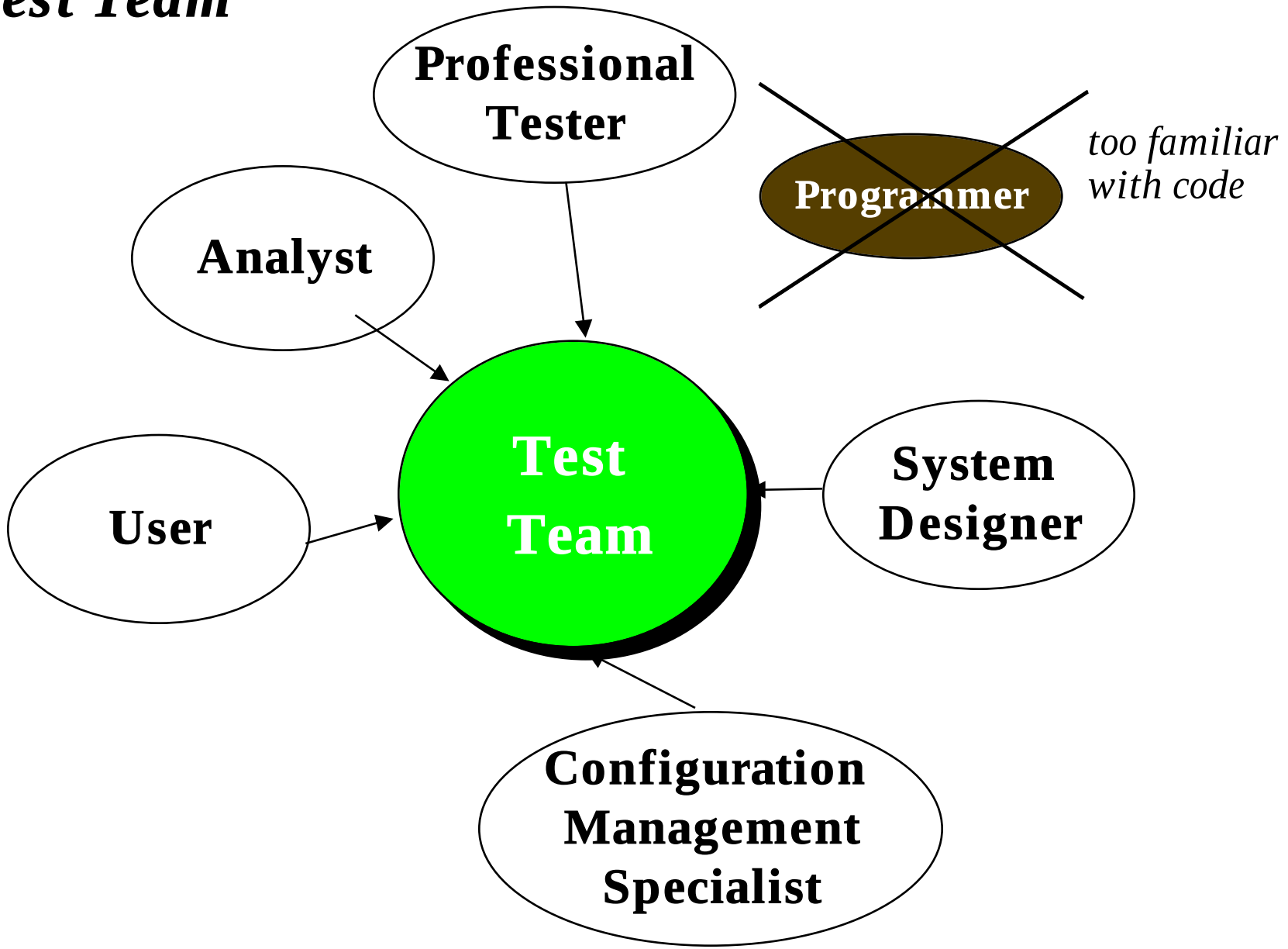
Evaluate the test results

Change system

Do regression testing



Test Team


Professional Tester
Programmer
too familiar with code
Analyst
Test Team
User
System Designer
Configuration Management Specialist

Summary

- Testing is still a black art, but many rules and heuristics are available
- Testing consists of unit testing, integration testing and system testing
- Testing has its own lifecycle
- Test documentation is crucial
- Test team: Different members with different backgrounds are important
- Issues not covered:
  - **Testing of parallel programs**
  - **Testing & configuration management**
  - **Testing multiple platforms**