INFORMATION SYSTEMS ANALYSIS AND DESIGN

Mr. JACKSON ALUNGA

jalungar@gmail.com

Overview

- Integration testing
  - Big bang
  - Bottom up
  - Top down
  - Sandwich
  - Continuous
- System testing
  - Functional
  - Performance
- Acceptance testing
- Summary

Integration Testing

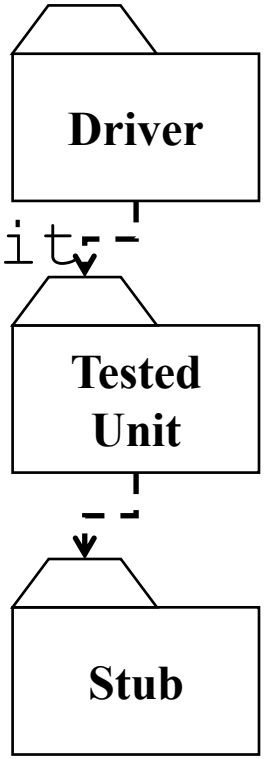
- The entire system is viewed as a collection of subsystems (sets of classes) determined during the system and object design
- Goal: Test all interfaces between subsystems and the interaction of subsystems
- The Integration testing strategy determines the order in which the subsystems are selected for testing and integration.

Why do we do integration testing?

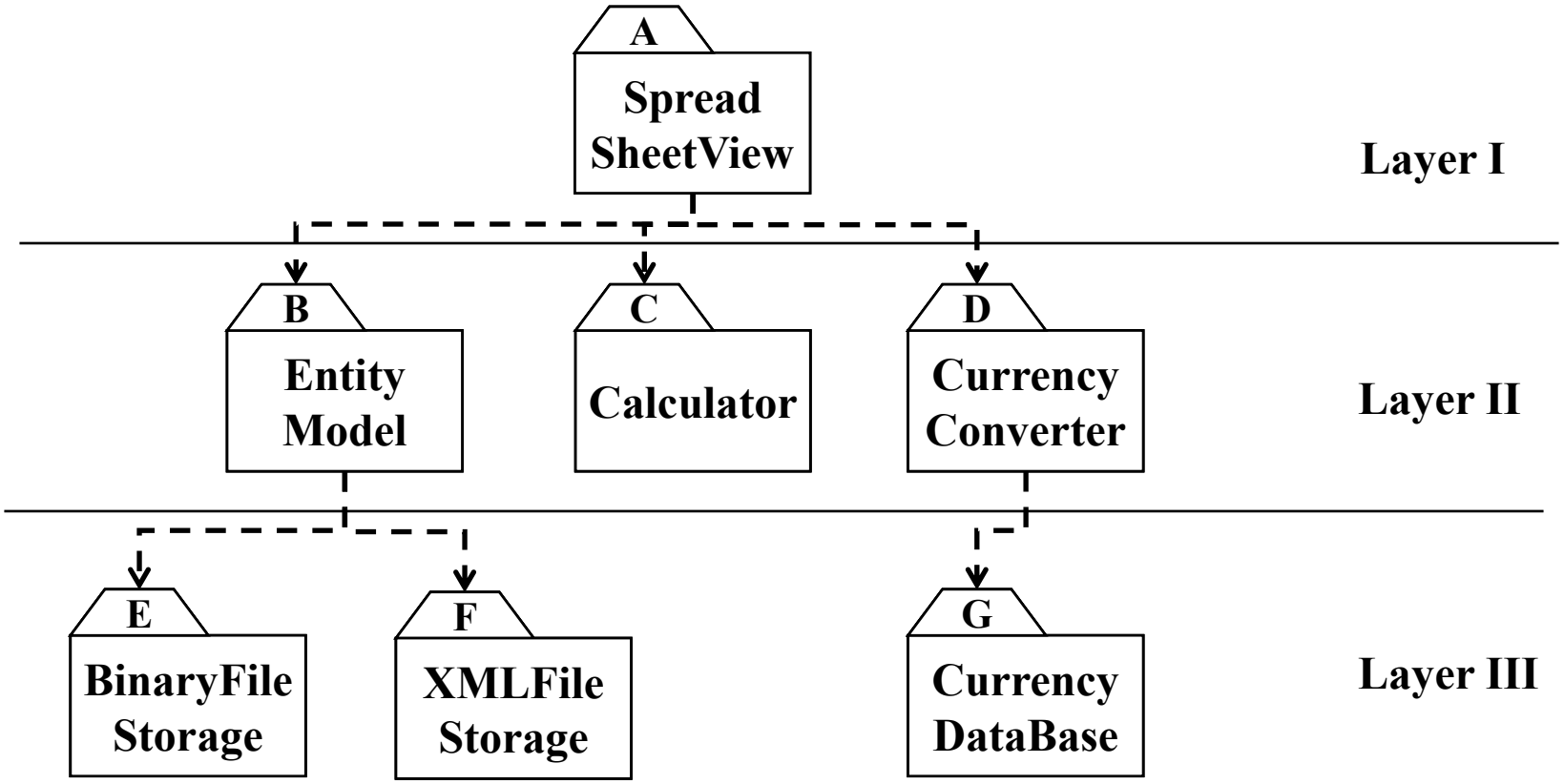
- Unit tests only test the unit in isolation
- Many failures result from faults in the interaction of subsystems
- Often many Off-the-shelf components are used that cannot be unit tested
- Without integration testing the system test will be very time consuming
- Failures that are not discovered in integration testing will be discovered after the system is deployed and can be very expensive.

Stubs and drivers

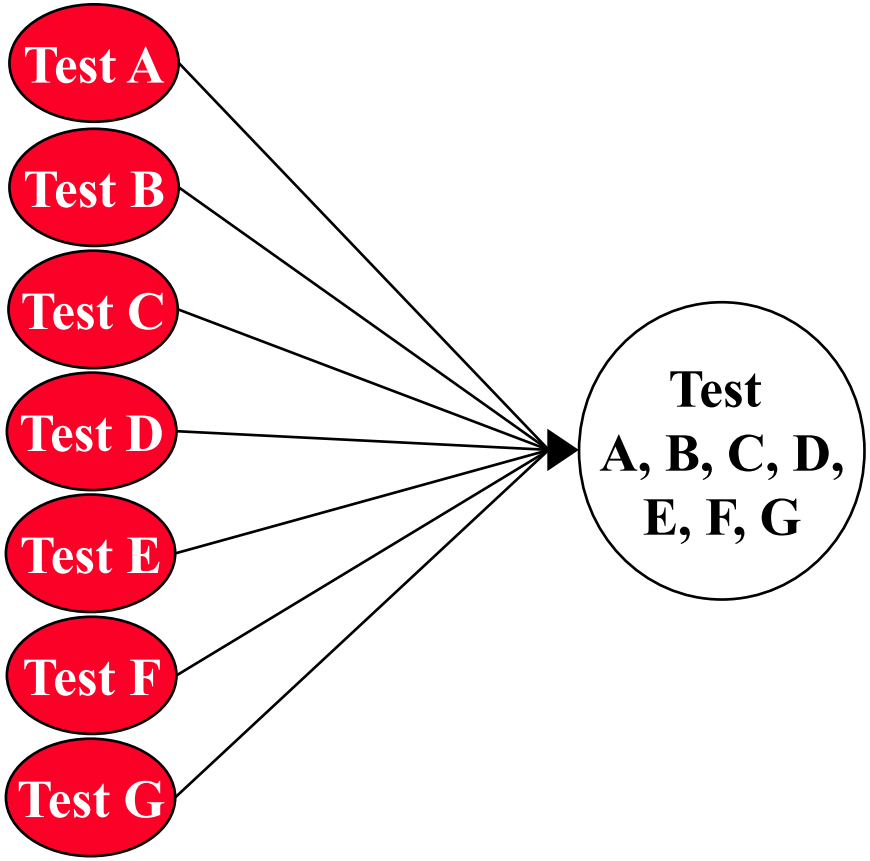
- Driver:
  - A component, that calls the `TestedUnit`
  - Controls the test cases
- Stub:
  - A component, the `TestedUnit` depends on
  - Partial implementation
  - Returns fake values.

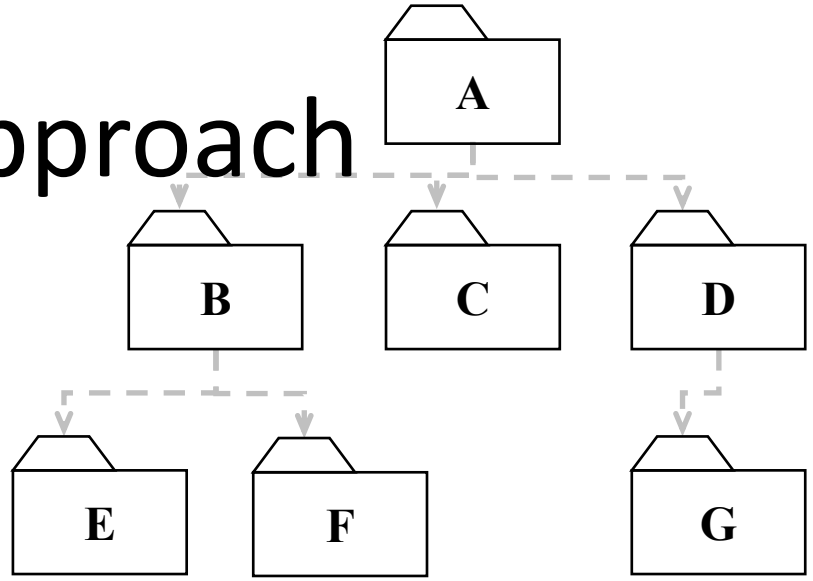


Example: A 3-Layer-Design (Spreadsheet)



Big-Bang Approach

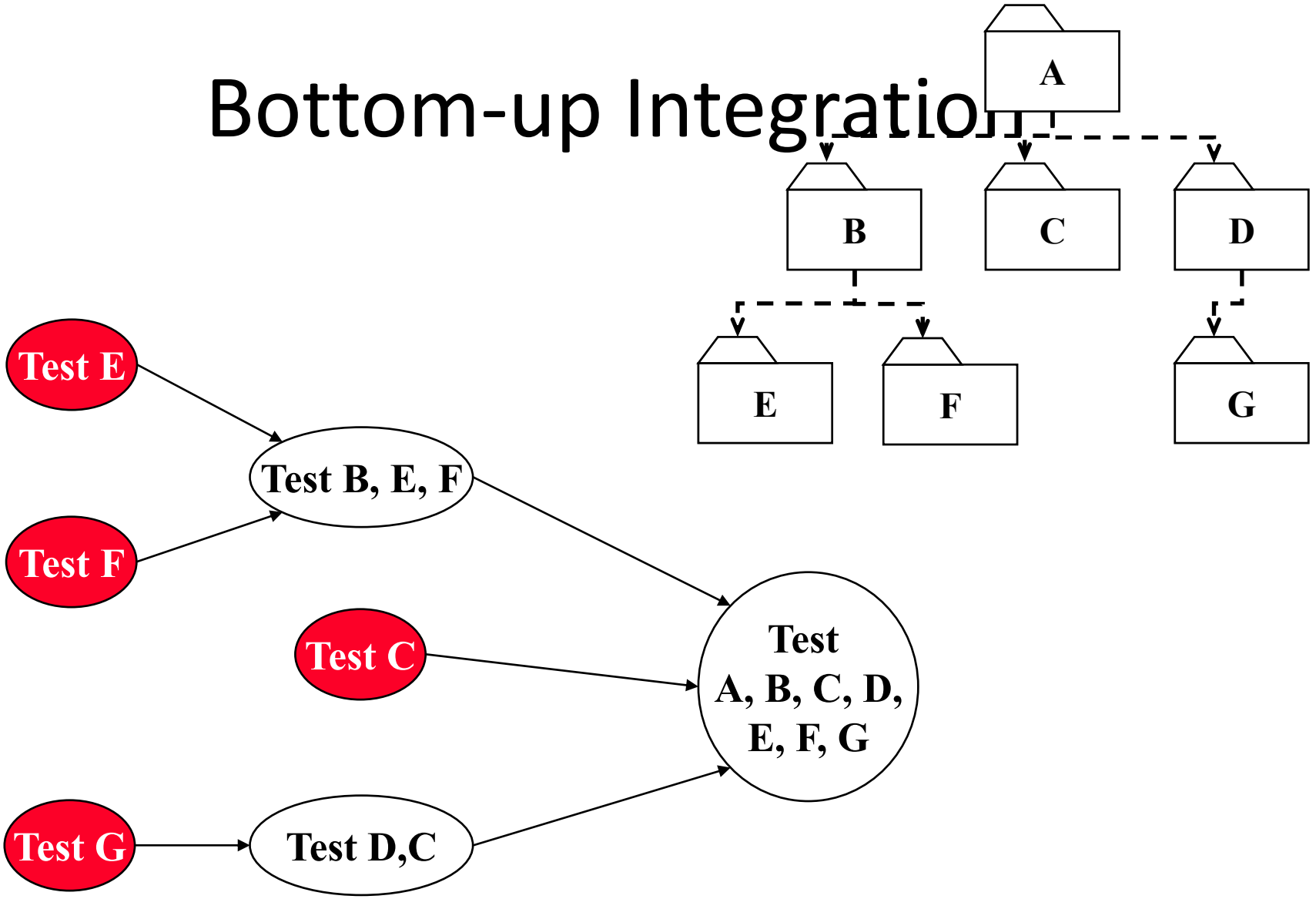




Bottom-up Testing Strategy

- The subsystems 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 repeated until all subsystems are included
- Drivers are needed.

Bottom-up Integration



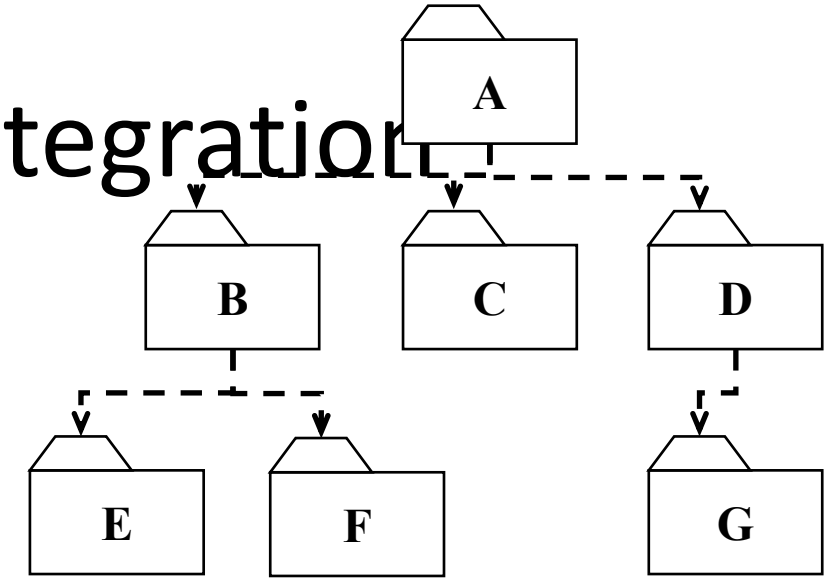
Pros and Cons of Bottom-Up Integration Testing

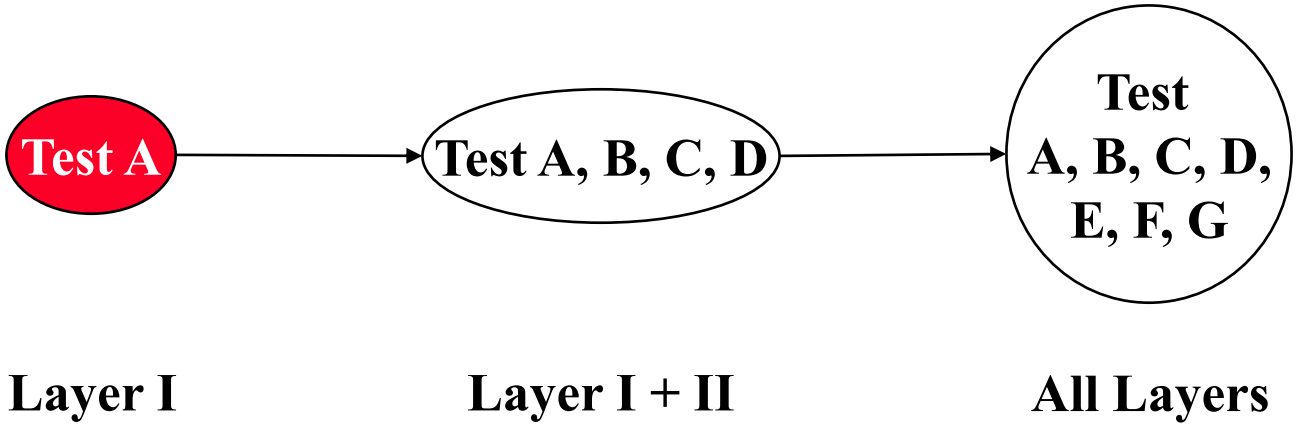
- Con:
  - Tests the most important subsystem (user interface) last
  - Drivers needed
- Pro
  - No stubs needed
  - Useful for integration testing of the following systems
    - Object-oriented systems
    - Real-time systems
    - Systems with strict performance requirements.

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
- Stubs are needed to do the testing.

Top-down Integration





Pros and Cons of Top-down Integration Testing

Pro

- Test cases can be defined in terms of the functionality of the system (functional requirements)
- No drivers needed

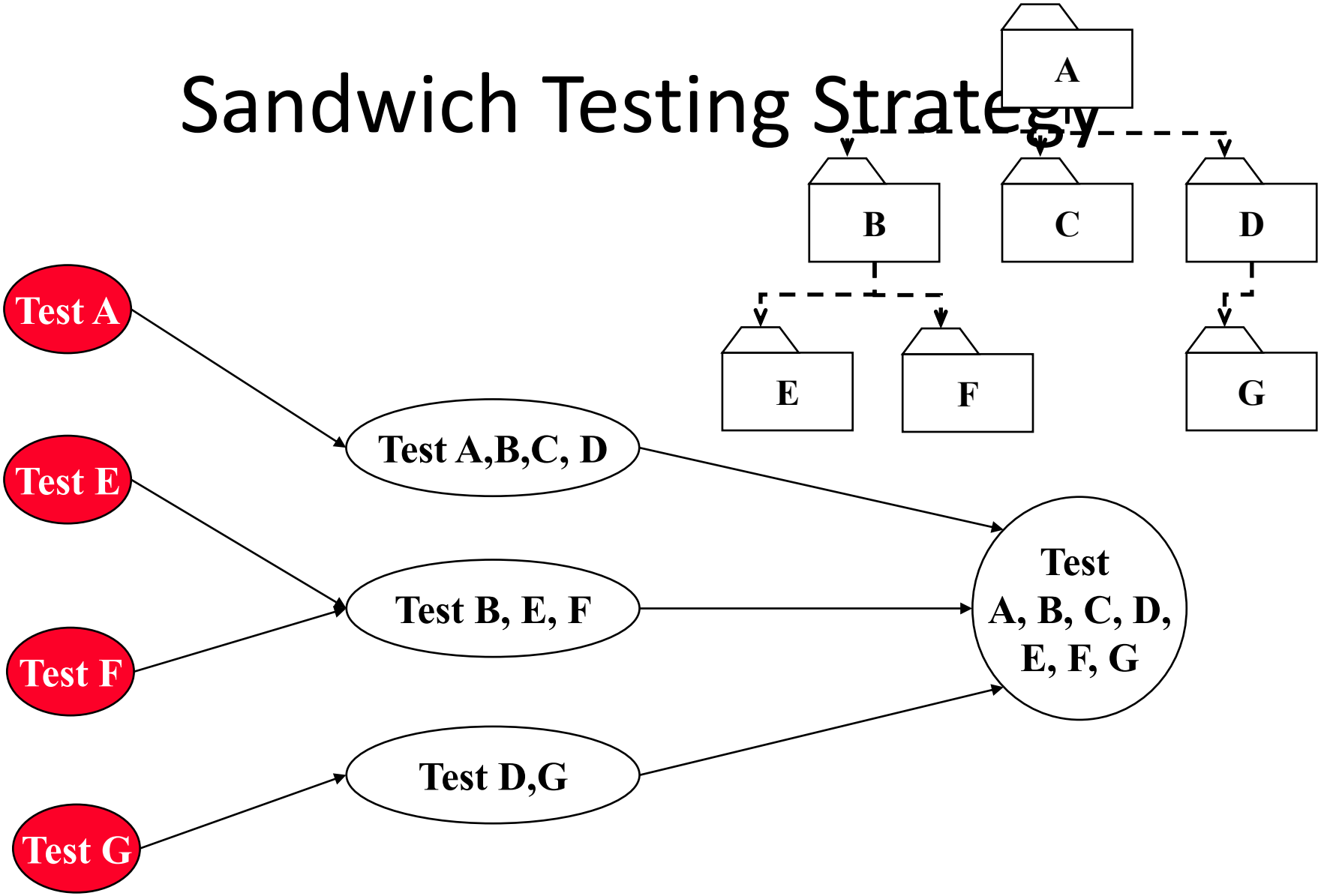
Cons

- Writing stubs is difficult: Stubs must allow all possible conditions to be tested.
- Large number of stubs may be required, especially if the lowest level of the system contains many methods.
- Some interfaces are not tested separately.

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.

Sandwich Testing Strategy



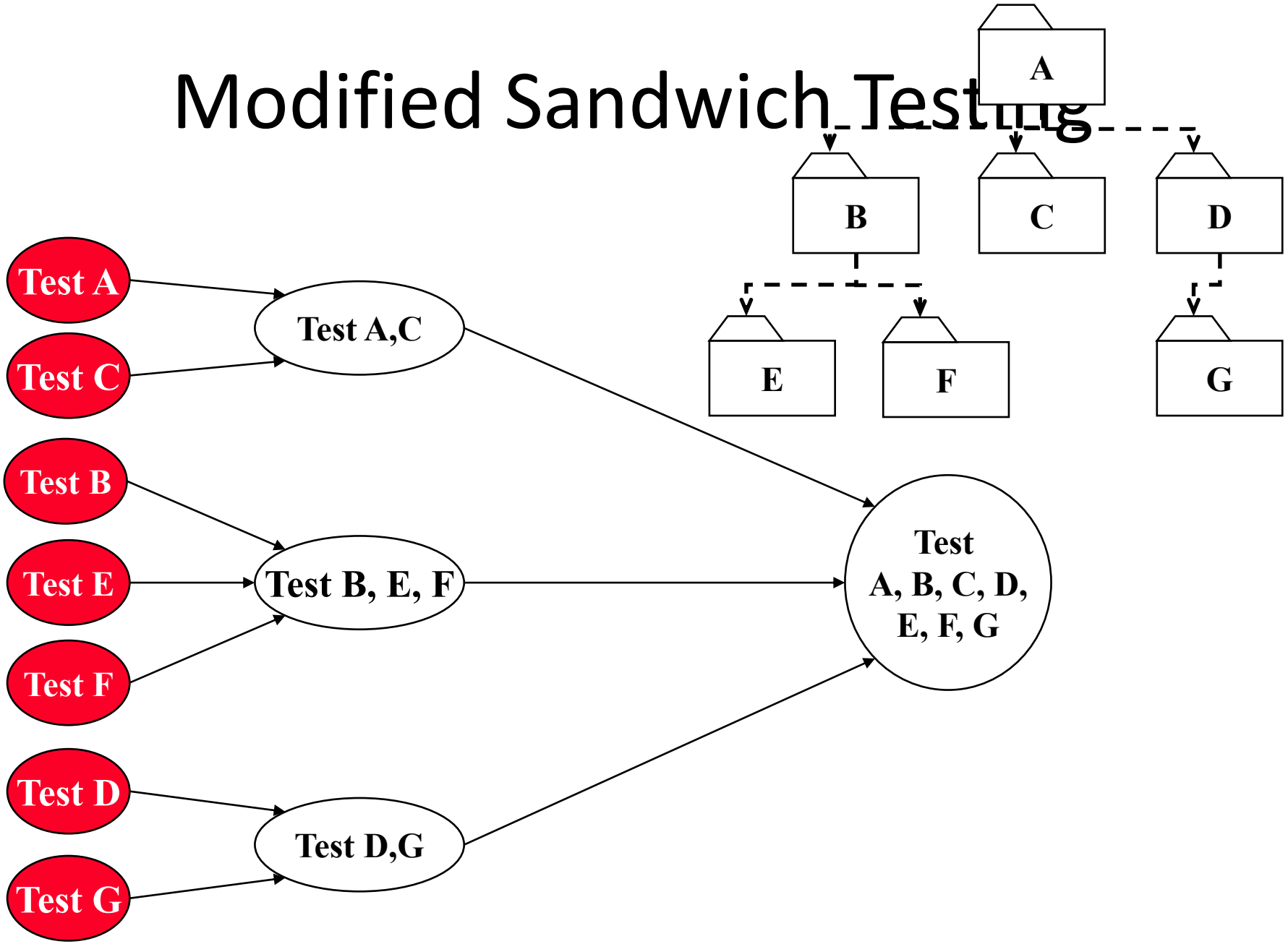
Pros and Cons of Sandwich Testing

- Top and Bottom Layer Tests can be done in parallel
- Problem: Does not test the individual subsystems and their interfaces 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



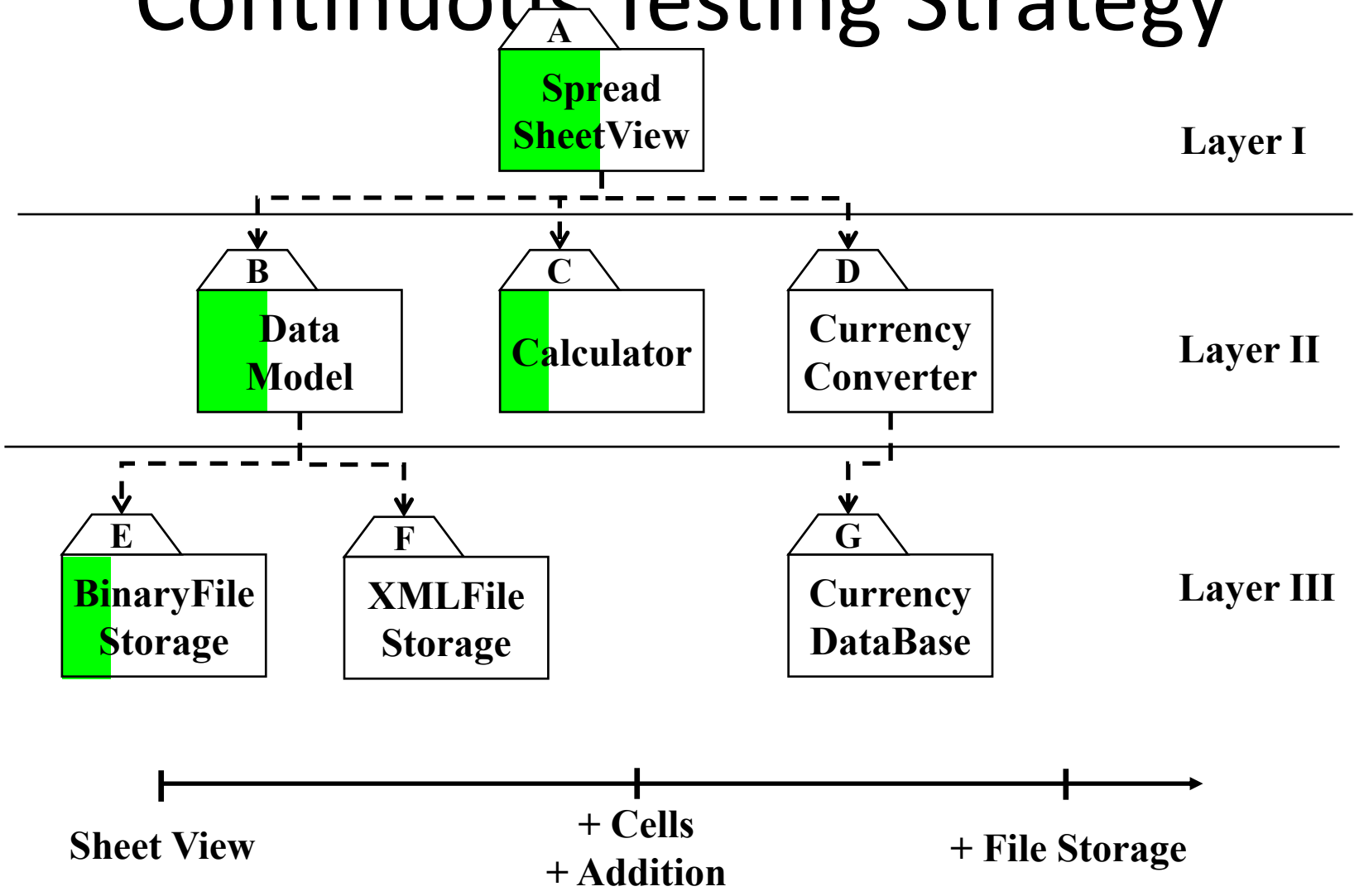
Continuous Testing

- Continuous build:
  - Build from day one
  - Test from day one
  - Integrate from day one

  $\Rightarrow$ System is always runnable

- Requires integrated tool support:
  - Continuous build server
  - Automated tests with high coverage
  - Tool supported refactoring
  - Software configuration management
  - Issue tracking.

Continuous Testing Strategy



Steps in Integration Testing

1. Based on the integration strategy, *select a component* to be tested. Unit test all the classes in the component.
2. Put selected component together; do any *preliminary fix-up* necessary to make the integration test operational (drivers, stubs)
3. Test functional requirements*:* Define test cases that exercise all uses cases with the selected component
4. Test subsystem decomposition*:* Define test cases that exercise all dependencies
5. Test non-functional requirements: Execute *performance tests*
6. *Keep records* of the test cases and testing activities.
7. Repeat steps 1 to 7 until the full system is tested.

The primary *goal of integration testing is to identify failures* with the (current) component *configuration*.

System Testing

- Functional Testing
  - Validates functional requirements. the system must perform functions specified in the requirements.
- Performance Testing
  - Validates non-functional requirements. the system must satisfy security, precision, load and speed constrains specified in the requirements
- Acceptance Testing
  - Validates clients expectations. Customers try the system (in the lab) to make sure that the system built is the system they requested.
- Deployment Testing
  - The software is deployed and tested in the production environment.

Functional Testing

Goal: Test functionality of system

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

Performance Testing

Goal: Try to violate non-functional requirements

- 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?

Types of Performance Testing

- Stress Testing
  - Stress limits of system
- Volume testing
  - Test what happens if large amounts of data are handled
- Configuration testing
  - Test the various software and hardware configurations
- Compatibility test
  - Test backward compatibility with existing systems
- Timing testing
  - Evaluate response times and time to perform a function
- Security testing
  - Try to violate security requirements
- Environmental test
  - Test tolerances for heat, humidity, motion
- Quality testing
  - Test reliability, maintain- ability & availability
- Recovery testing
  - Test system's response to presence of errors or loss of data
- Human factors testing
  - Test with end users.

Acceptance Testing

- Goal: Demonstrate system is ready for operational use
  - Choice of tests is made by client
  - Many tests can be taken from integration testing
  - Acceptance test is performed by the client, not by the developer.

- Alpha test:
  - Client uses the software at the developer's environment.
  - Software used in a controlled setting, with the developer always ready to fix bugs.
- Beta test:
  - Conducted at client's environment (developer is not present)
  - Software gets a realistic workout in target environ- ment

Acceptance Testing

The purpose is to enable customers and users to determine if the system built really meets their needs and expectations.

Benchmarking: a predetermined set of test cases corresponding to typical usage conditions is executed against the system

Pilot Testing: users employ the software as a small-scale experiment or in a controlled environment

Alpha-Testing: pre-release closed / in-house user testing

Beta-Testing: pre-release public user testing

Parallel Testing: old and new software are used together and the old software is gradually phased out.

Acceptance testing uncovers requirement discrepancies as well as helps users to find out what they really want (hopefully not at the developer's expense!)

Deployment Testing

The software is installed on a target system and tested with:

- Various hardware configurations
- Various supported OS versions and service packs
- Various versions of third-party components (e.g. database servers, web servers, etc.)
- Various configurations of resident software

Testing has many activities

Establish the test objectives

Design the test cases

Write the test cases

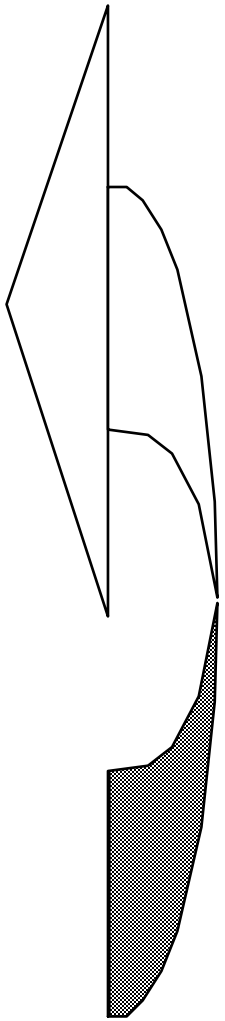
Test the test cases

Execute the tests

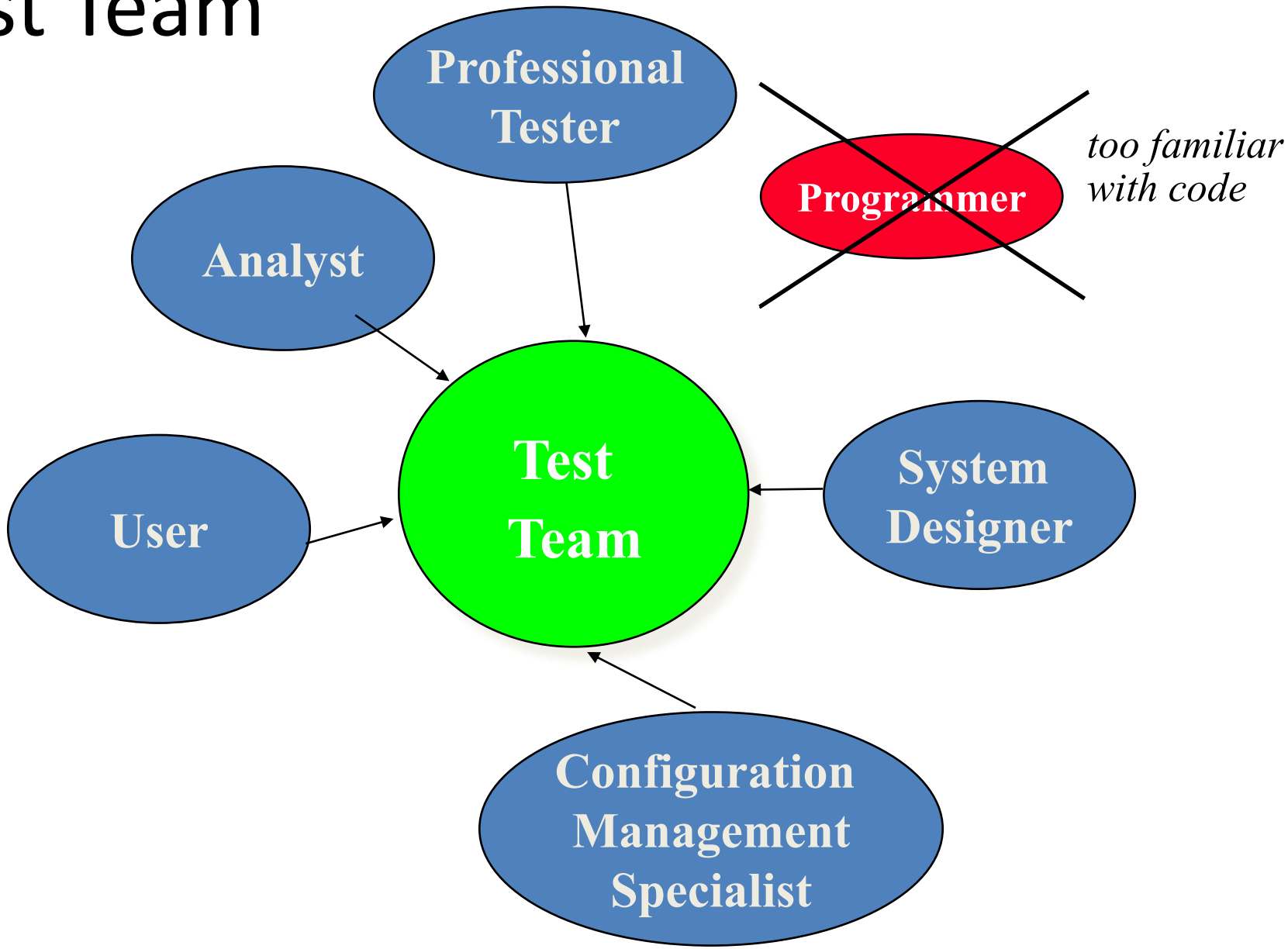
Evaluate the test results

Change the system

Do regression testing



Test Team


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

The 4 Testing Steps

1. Select what has to be tested
    - Analysis: Completeness of requirements
    - Design: Cohesion
    - Implementation: Source code
2. Decide how the testing is done
    - Review or code inspection
    - Proofs (Design by Contract)
    - Black-box, white box,
    - Select integration testing strategy (big bang, bottom up, top down, sandwich)

3. Develop test cases
    - A test case is a set of test data or situations that will be used to exercise the unit (class, subsystem, system) being tested or about the attribute being measured
4. Create the test oracle
    - An oracle contains the predicted results for a set of test cases
    - The test oracle has to be written down before the actual testing takes place.

Guidance for Test Case Selection

- Use *analysis knowledge* about functional requirements (black-box testing):
  - Use cases
  - Expected input data
  - Invalid input data
- Use *design knowledge* about system structure, algorithms, data structures (white-box testing):
  - Control structures
    - Test branches, loops, ...
  - Data structures
    - Test records fields, arrays, ...

- Use *implementation knowledge* about algorithms and datastructures:
  - Force a division by zero
  - If the upper bound of an array is 10, then use 11 as index.

Safety-Critical Systems

Safe means free from accident or loss.

Hazard: a system state that, together with the right conditions, can lead to an accident.

Failure Mode: a situation that can lead to a hazard.

We can build a fault tree to trace known failure modes to unknown effects / hazards.

Safety-Critical System Issues

Remember Murphy's Laws:

- If a fault can occur it will
- If a user can make a mistake he will
- Least probable fault causes are not

* 100% reliable system is not necessarily safe or secure!
* Budget with testing in mind.

Hazard and Operability Studies

HAZOPS involves structured analysis to anticipate system hazards and to suggest means to avoid or deal with them.

During testing we must select test cases to exercise each failure mode to make sure that the system reacts in a safe manner.

Cleanroom

Method devised by IBM for developing high-quality software with a high-productivity team.

Principles:

1) Software must be *certified* with respect to the specifications (rather than tested)
2) Software must be zero-fault.

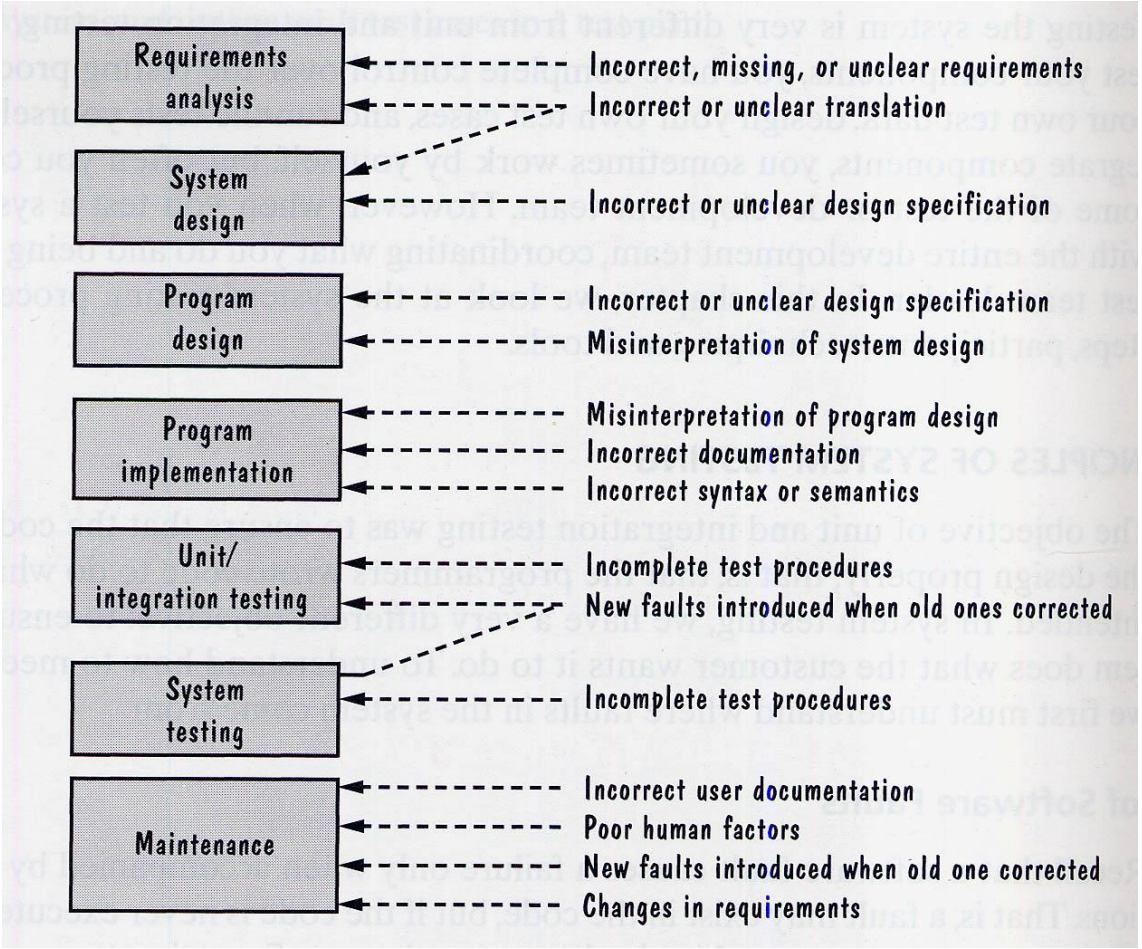
Cleanroom Process

1) The software as specified as a black box, then refined as a state box, then refined as a clear box.
2) The box structure encourages analysts to find flaws and omissions early in the project life cycle (when it is cheaper and faster to fix them).
3) The clear-box specification is converted into an intended function using natural language or mathematic notation.
4) For each function a correctness theorem is devised and proven.

* Unit testing is not necessary or even permitted!
* Errors found by statistical testing tends to be simple mistakes that are easy to fix as the cleanroom eliminates deep principle problems.

IBM claims an order of magnitude fault reduction (e.g. 3 faults per 1000 lines of code).

Causes of Software Faults



Summary

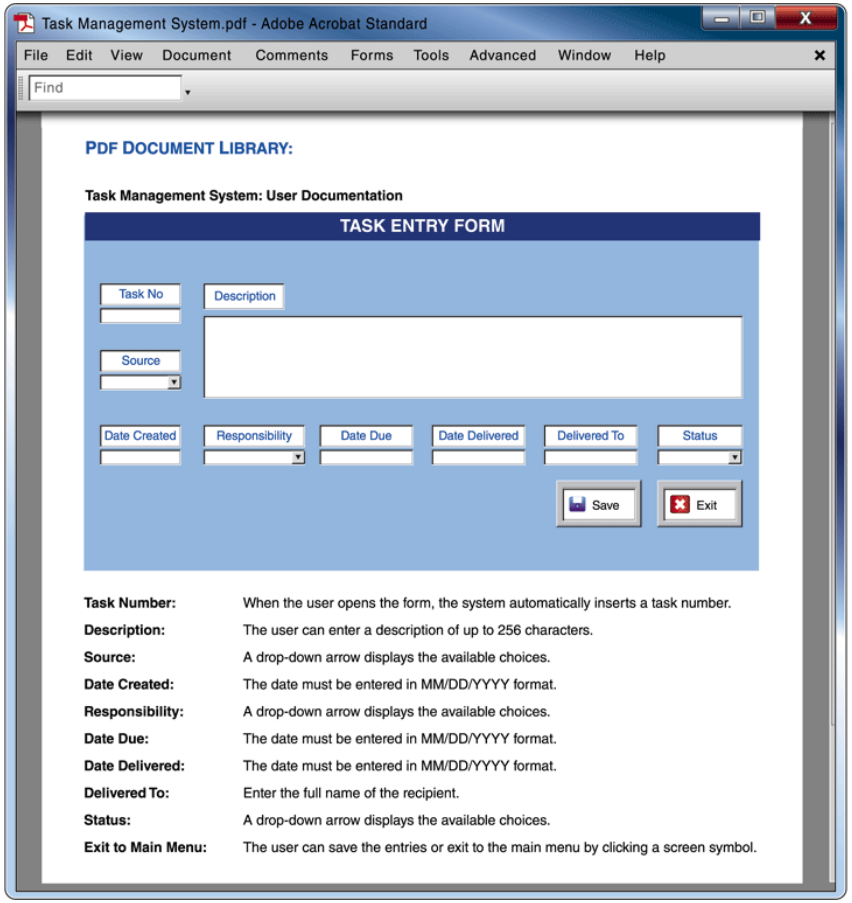
- Testing is still a black art, but many rules and heuristics are available
- Testing consists of
  - Unit testing
  - Integration testing
  - System testing
  - Acceptance testing
- Design patterns can be used for integration testing
- Testing has its own lifecycle

Documentation

- Program Documentation
- System Documentation
- Operations Documentation
- User Documentation
  - Systems analysts usually are responsible for preparing documentation to help users learn the system

Documentation

- User Documentation
  - Effective online documentation is an important productivity tool
  - Written documentation material also is valuable



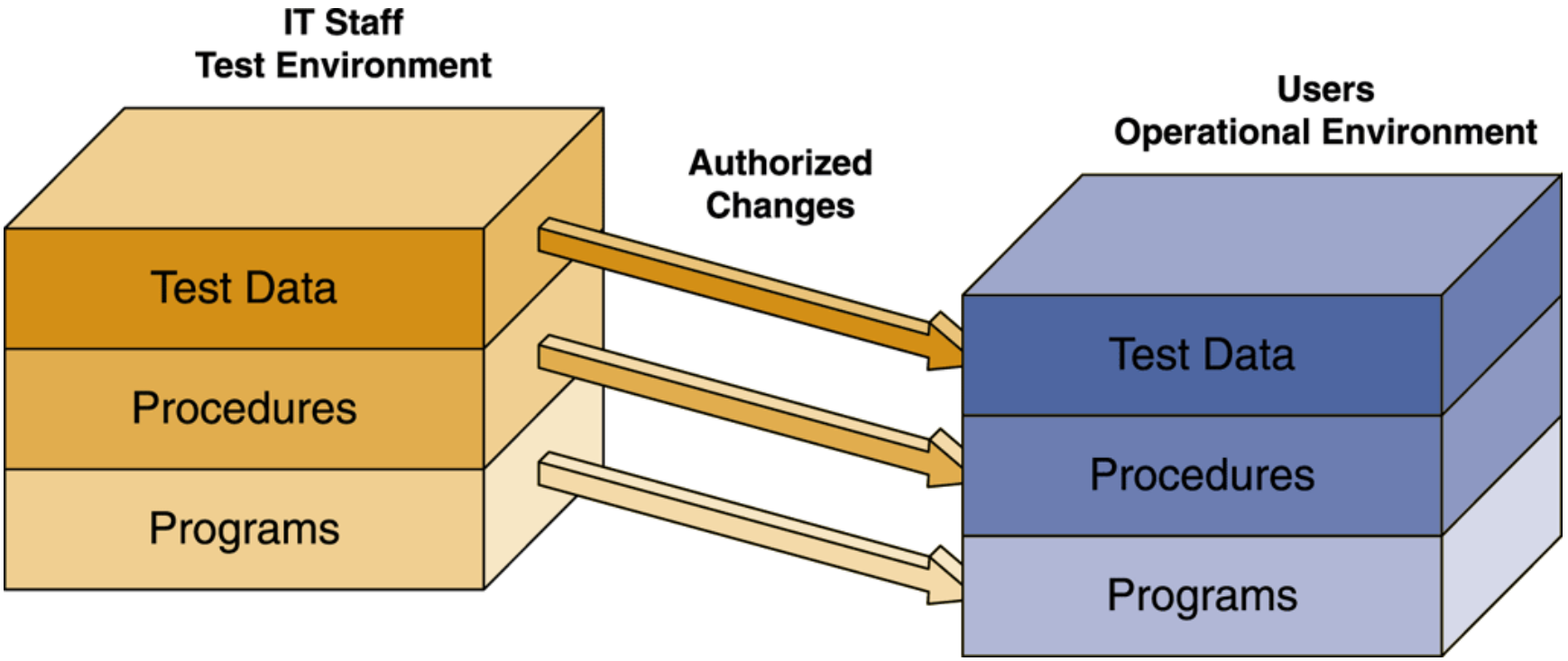
Management Approval

- After system testing is complete, you present the results to management
- If system testing produced no technical, economical, or operational problems, management determines a schedule for system installation and evaluation

System Installation and Evaluation

- Remaining steps in systems implementation:
  - Prepare a separate operational and test environment
  - Provide training for users, managers, and IT staff
  - Perform data conversion and system changeover
  - Carry out post-implementation evaluation of the system
  - Present a final report to management

Operational and Test Environments

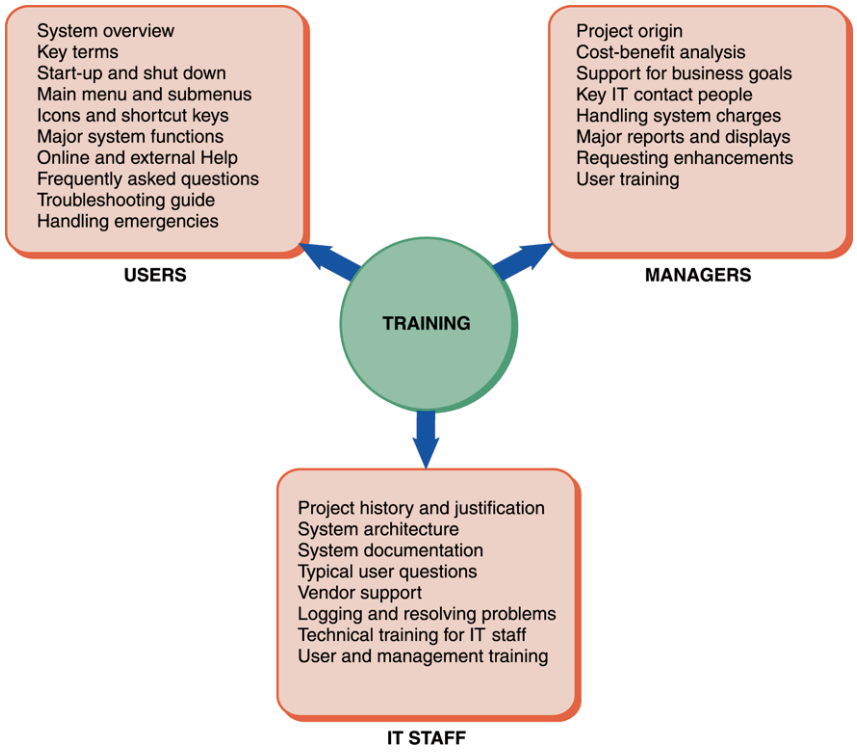


Operational and Test Environments

- The operational environment includes hardware and software configurations and settings, system utilities, telecommunications resources, and any other components that might affect system performance
- If you have to build or upgrade network resources to support the new system, you must test the platform rigorously before system installation begins

Training

- Training Plan
  - The three main groups for training are users, managers, and IT staff
  - You must determine how the company will provide training
- Vendor Training
  - Often gives the best return on your training dollars



Training

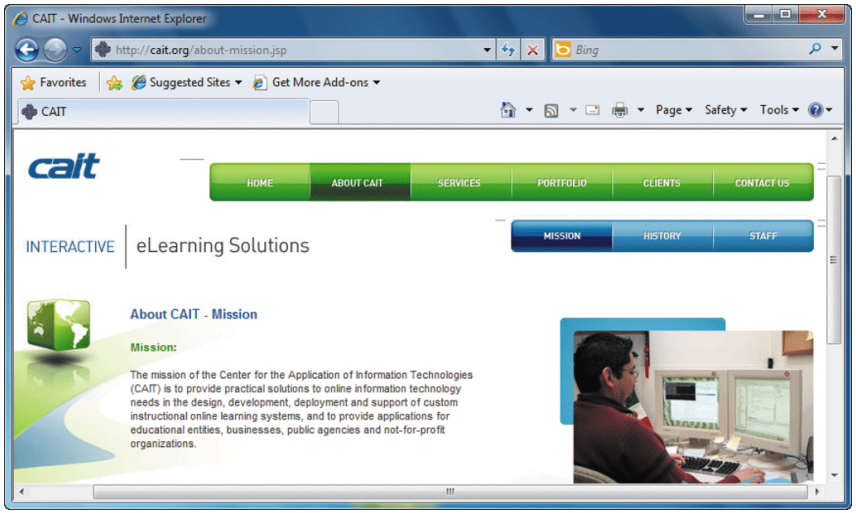
- Vendor Training
  - If the system includes the purchase of software or hardware, then vendor-supplied training is one of the features you should investigate in the RFPs (requests for proposal) and RFQs (requests for quotation) that you send to potential vendors
  - Often gives the best return on your training dollars

Training

- Webinars, Podcasts, and Tutorials
  - Webcast
  - Subscribers
  - As technology continues to advance, other wireless devices such as PDAs and cell phones will be able to receive podcasts
  - Tutorials can be developed by software vendors, or by a company's IT team

Training

- Outside Training Resources
  - Many training consultants, institutes, and firms are available that provide either standardized or customized training packages



Training

- Training Tips
  - Train people in groups, with separate training programs for distinct groups
  - Select the most effective place to conduct the training
  - Provide for learning by hearing, seeing, and doing
  - Prepare effective training materials, including interactive tutorials
  - Rely on previous trainees

Training

- Interactive Training
  - Usually, a relationship exists between training methods and costs
  - Online training
    - Should include step-by-step instructions
  - Video tutorials
    - You don’t have to be a professional video developer to create effective training tutorials

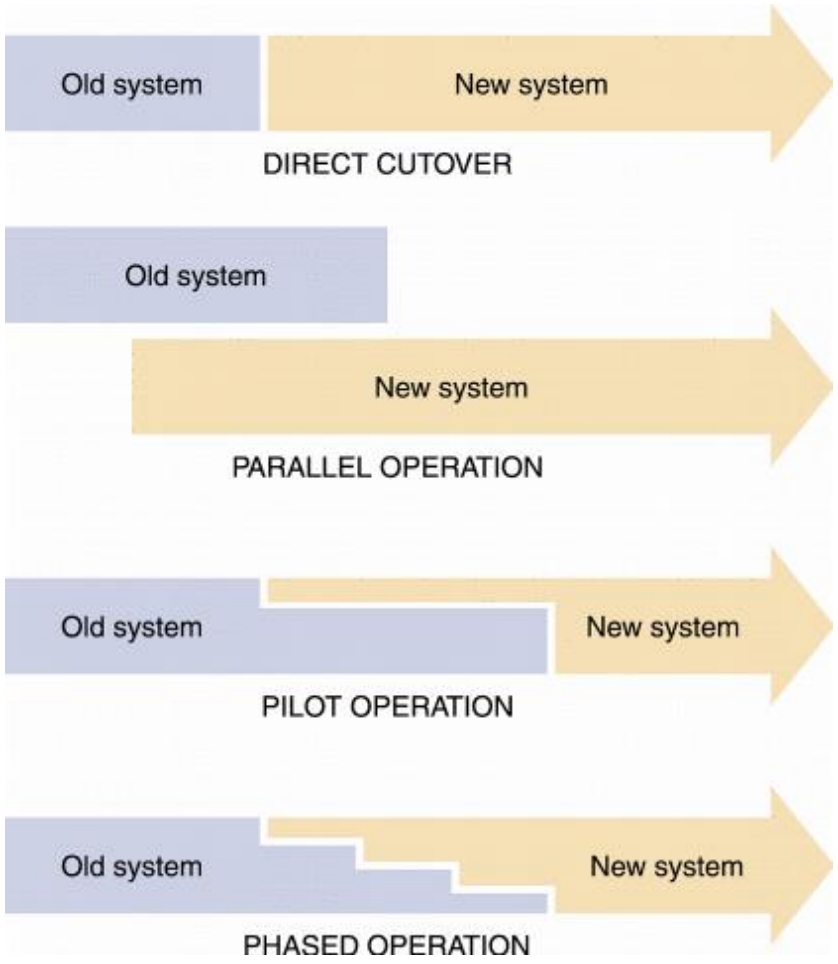
Data Conversion

- Data Conversion Strategies
  - The old system might be capable of exporting data in an acceptable format for the new system or in a standard format such as ASCII or ODBC
  - If a standard format is not available, you must develop a program to extract the data and convert it
  - Often requires additional data items, which might require manual entry

Data Conversion

- Data Conversion Security and Controls
  - You must ensure that all system control measures are in place and operational to protect data from unauthorized access and to help prevent erroneous input
  - Some errors will occur
  - It is essential that the new system be loaded with accurate, error-free data

System Changeover



System Changeover

- Direct Cutover
  - Involves more risk than other changeover methods
  - Companies often choose the direct cutover method for implementing commercial software packages
  - Cyclical information systems usually are converted using the direct cutover method at the beginning of a quarter, calendar year, or fiscal year

System Changeover

- Parallel Operation
  - Easier to verify that the new system is working properly under parallel operation than under direct cutover
  - Running both systems might place a burden on the operating environment and cause processing delay
  - Is not practical if the old and new systems are incompatible technically
  - Also is inappropriate when the two systems perform different functions

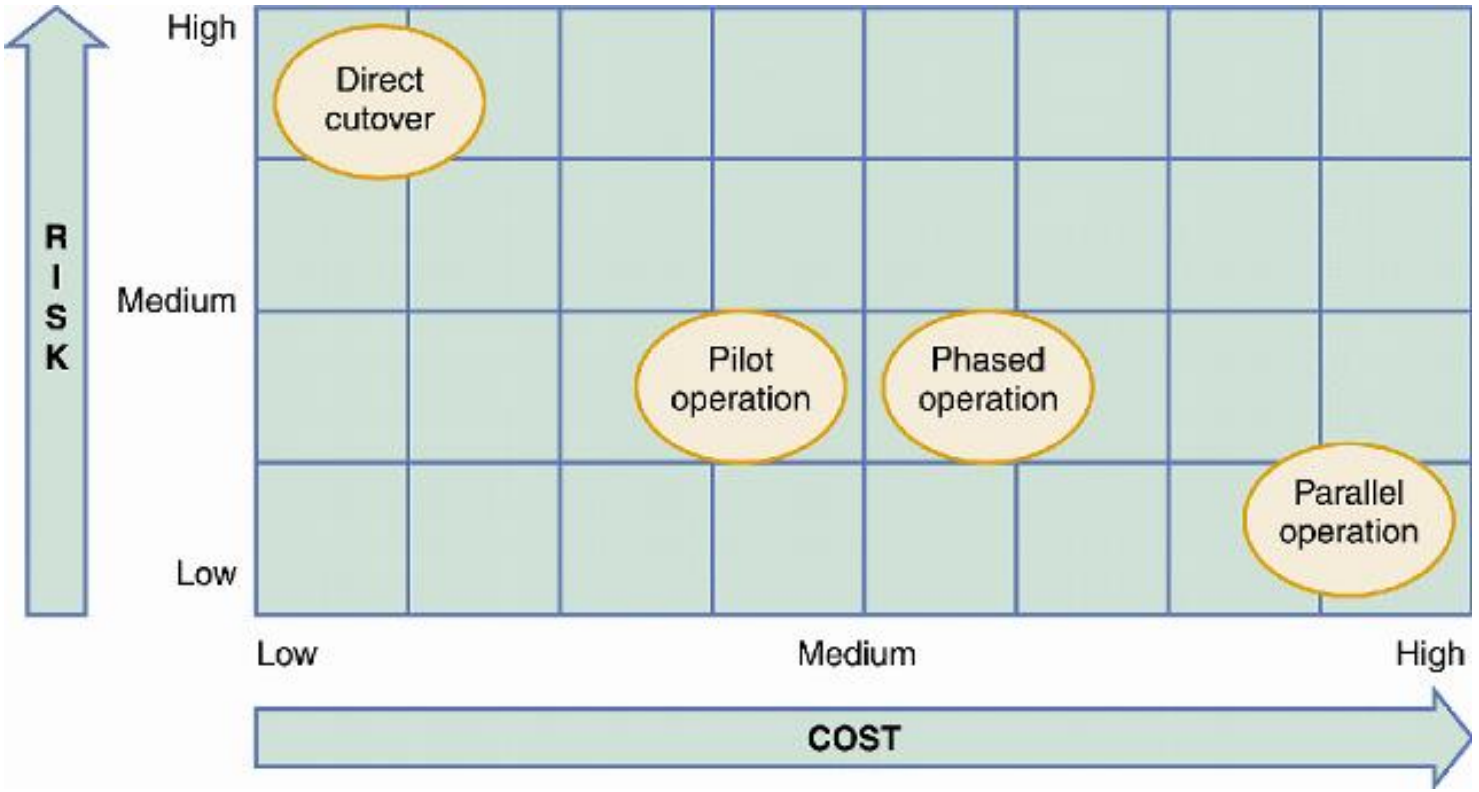
System Changeover

- Pilot Operation
  - The group that uses the new system first is called the pilot site
  - The old system continues to operate for the entire organization
  - After the system proves successful at the pilot site, it is implemented in the rest of the organization, usually using the direct cutover method
  - Is a combination of parallel operation and direct cutover methods

System Changeover

- Phased Operation
  - You give a part of the system to all users
  - The risk of errors or failures is limited to the implemented module only
  - Is less expensive than full parallel operation
  - Is not possible, however, if the system cannot be separated easily into logical modules or segments

System Changeover



Post-Implementation Tasks

- Post-Implementation Evaluation
  - A post-implementation evaluation should examine all aspects of the development effort and the end product — the developed information system
  - You can apply the same fact-finding techniques in a post-implementation evaluation that you used to determine the system requirements during the systems analysis phase

Post-Implementation Tasks

- Final Report to Management
  - Your report should include the following:
    - Final versions of all system documentation
    - Planned modifications and enhancements to the system that have been identified
    - Recap of all systems development costs and schedules

Post-Implementation Tasks

- Final Report to Management
  - Your report should include the following:
    - Comparison of actual costs and schedules to the original estimates
    - Post-implementation evaluation, if it has been performed
  - Marks the end of systems development work