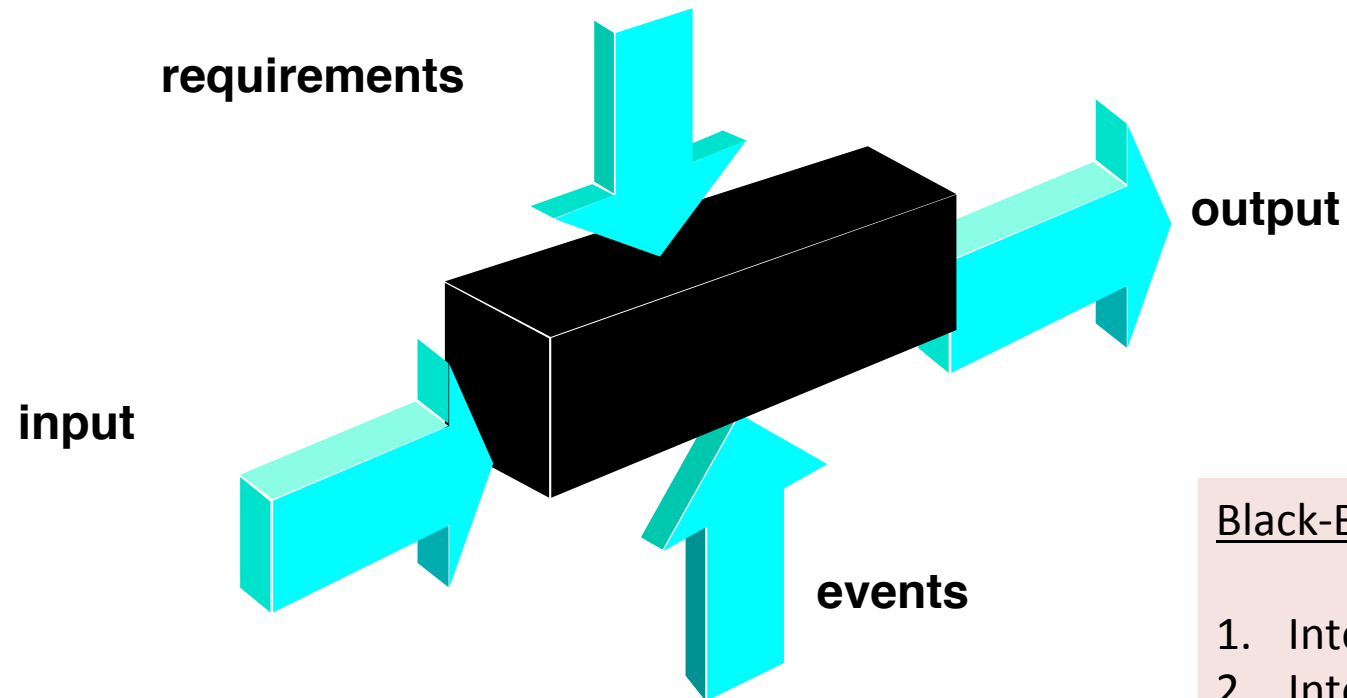


CMPS411
Spring 2018

Lecture 13

Black-Box Testing

Black-Box Testing



Black-Box testing

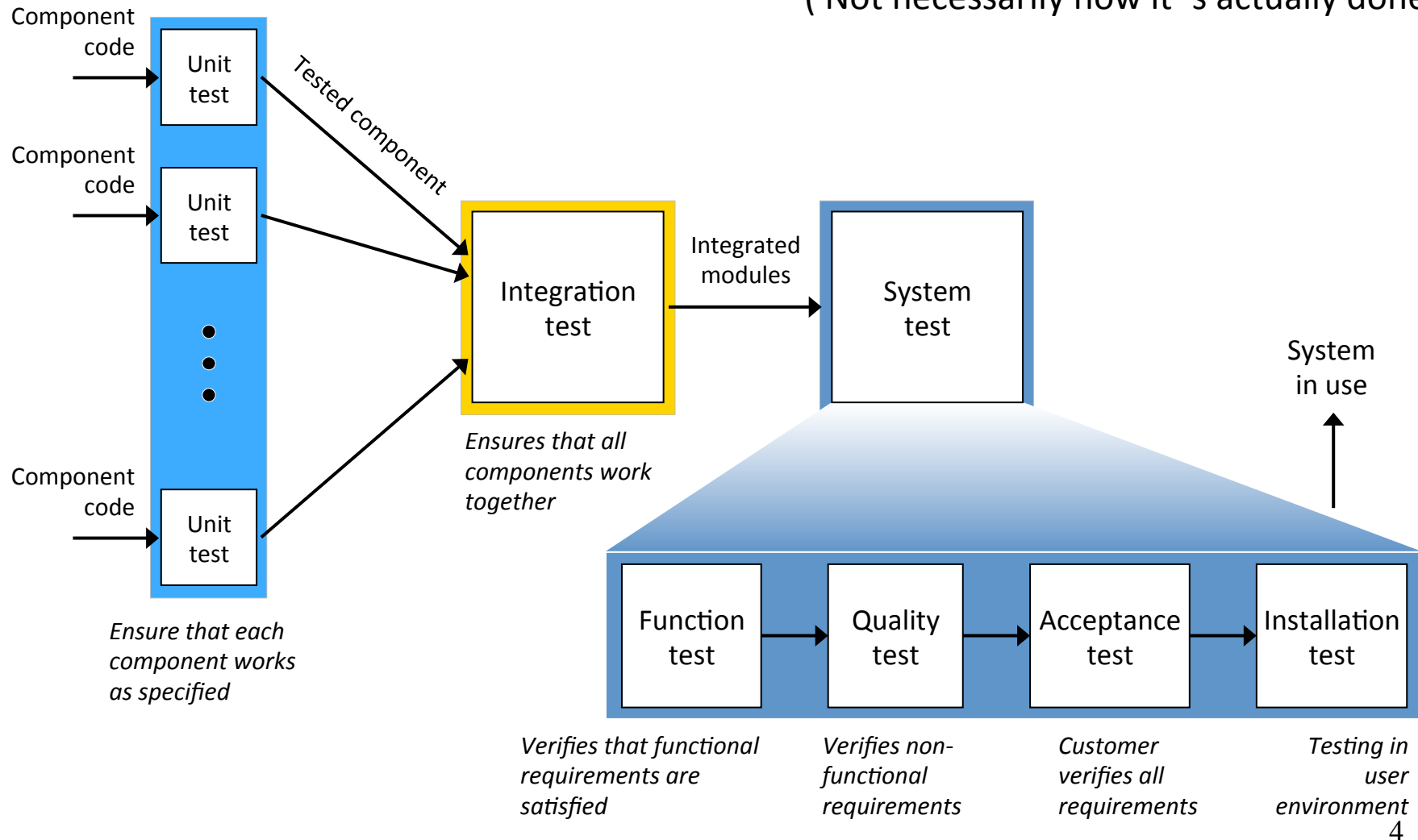
1. Integration testing
2. Interface testing
3. System testing
4. Use case testing
5. Release testing
6. User testing
7. Accepting testing

Black-Box Testing

- ✧ How is functional validity tested?
- ✧ How is system behavior and performance tested?
- ✧ What classes of input will make good test cases?
- ✧ Is the system particularly sensitive to certain input values?
- ✧ How are the boundaries of a data class isolated?
- ✧ What data rates and data volume can the system tolerate?
- ✧ What effect will specific combinations of data have on system operation?

Logical Organization of Testing

(Not necessarily how it's actually done!)



Types of Black-Box Testing

✧ Unit Testing:

- Individual *subsystem*
- Carried out by developers
- Goal: Confirm that individual subsystem/module is correctly coded and carries out the intended functionality

✧ Integration Testing:

- Groups of subsystems (collection of classes) and eventually the entire system
- Carried out by developers
- Goal: Test the *interface* among the subsystem

System Testing

✧ **System** Testing (Functional test and Performance test):

- The entire system
- Carried out by developers
- Goal: Determine if the system meets the *requirements* (functional and *non functional*)

✧ **Acceptance** Testing and **Installation** Testing:

- Evaluates the system delivered by developers
- Carried out by the *client*. May involve executing typical transactions on site on a trial basis
- Goal: Demonstrate that the system meets customer *requirements* and is ready to use

Unit Testing

✧ Informal:

- Incremental coding

Write a little, test a little

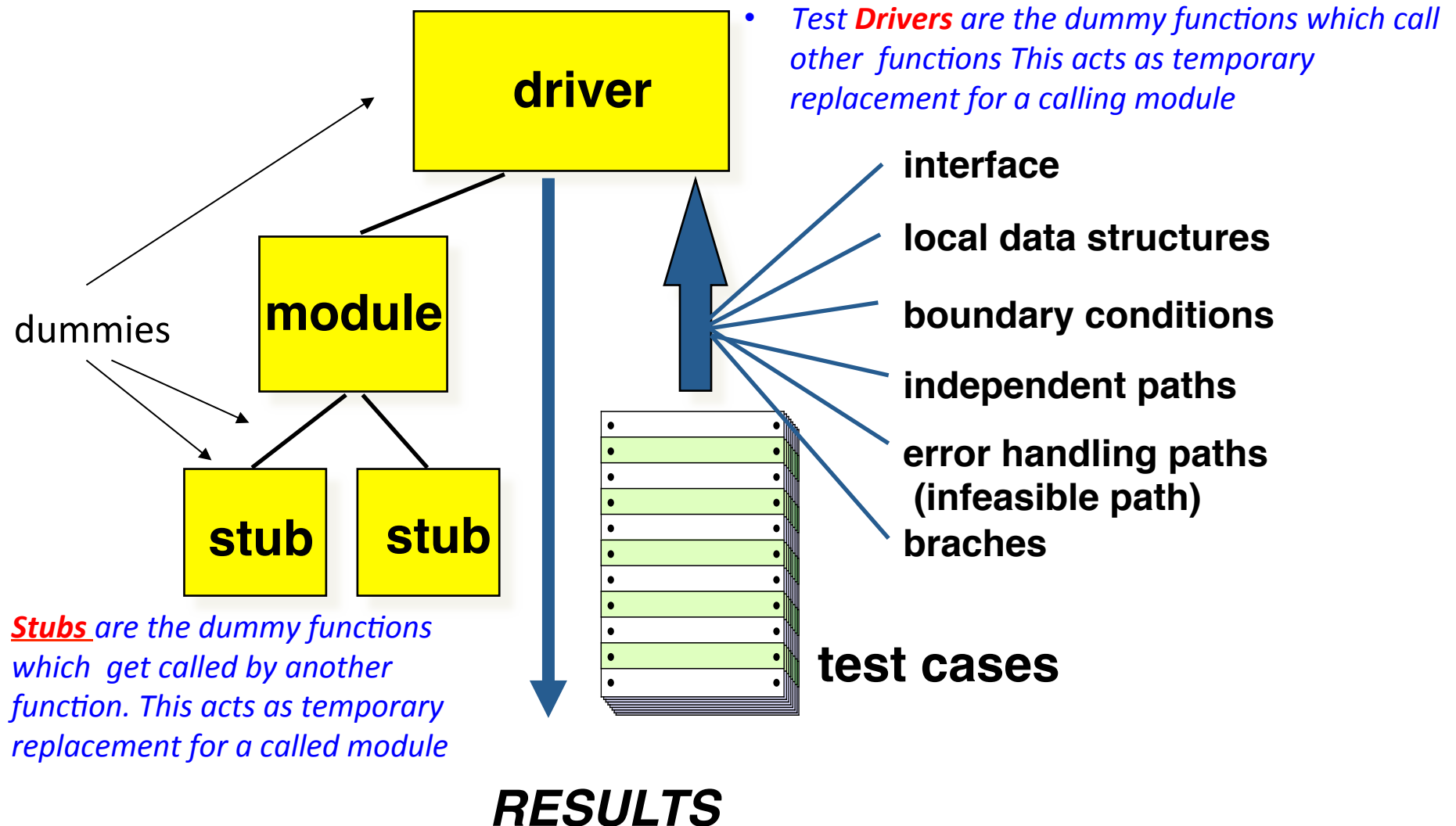
✧ Static Analysis:

- Hand execution: Reading the *source code*
- Walk-Through (informal presentation to others)
- Code Inspection (formal presentation to others)
- Automated Tools, checking for
 - syntactic and semantic errors
 - departure from coding standards

✧ Dynamic Analysis:

- *White-box* testing (Test the internal logic of the subsystem or object)
- Black-box testing (Test the input/output behavior)
- Data-structure based testing (Data types determine test cases)

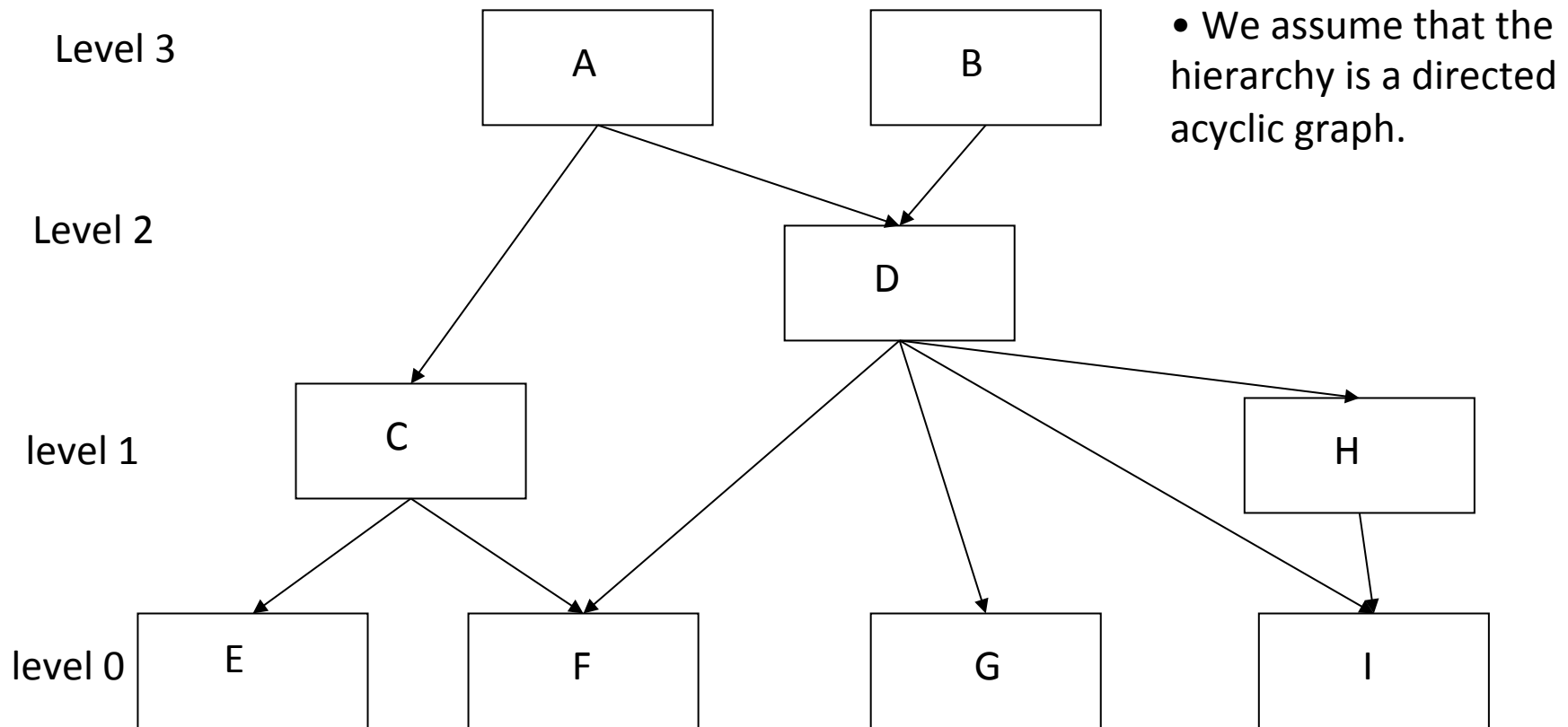
Unit Test Environment



Integration Testing

- ✧ Integration testing: Integrated collection of modules tested as a group or partial system
- ✧ Integration plan specifies the order in which to combine modules into partial systems
- ✧ Different approaches to integration testing
 - Bottom-up
 - Top-down
 - Big-bang
 - Sandwich
- ✧ **Stubs** are used during Top-down integration testing, in order to simulate the behaviour of the lower-level modules that are not yet integrated or developed.
- ✧ **Stubs** are the modules that act as temporary replacement for a called module and give the same output as that of the actual product.
- ✧ **Drivers** are used during Bottom-up integrating testing.

Module Structure



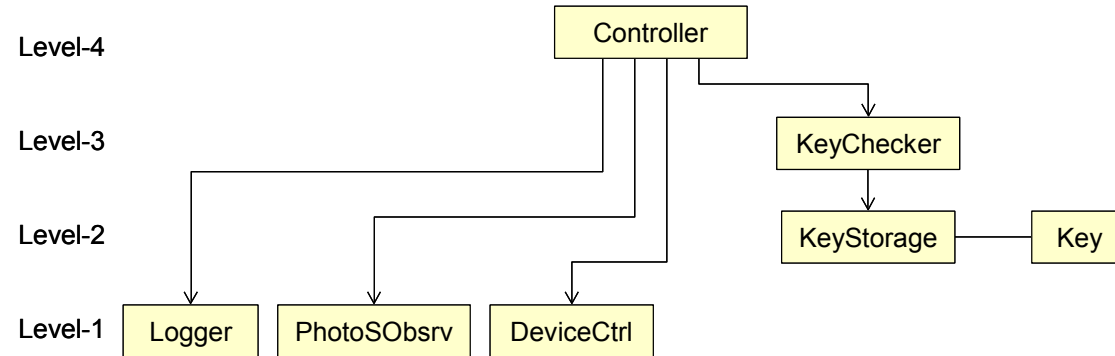
- A uses C and D; B uses D; C uses E and F; D uses F, G, H and I; H uses I
- Modules A and B are at level 3; Module D is at level 2

Modules C and H are at level 1; Modules E, F, G, I are at level 0

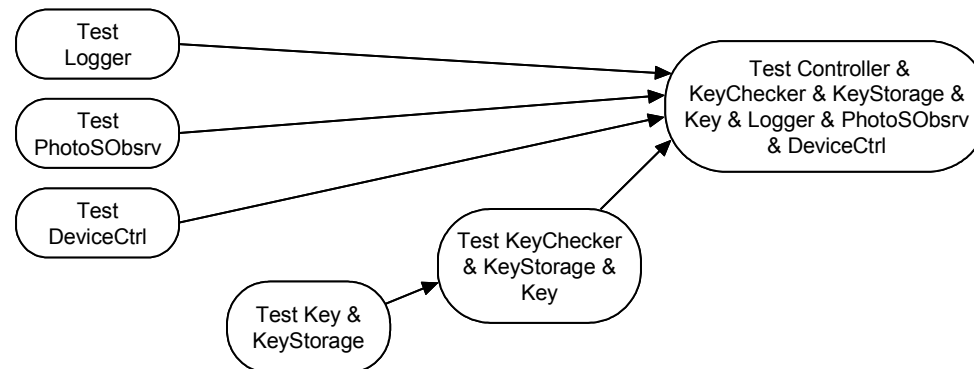
- level 0 components do not use any other components
- level i components use at least one component on level $i-1$ and no component at a level higher than $i-1$

Horizontal Integration Testing

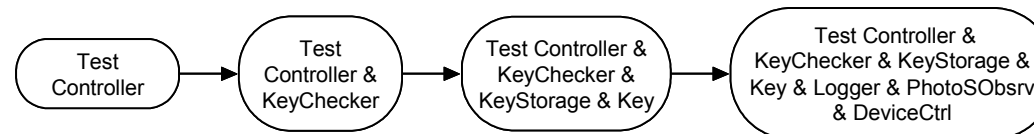
System hierarchy:



Bottom-up integration testing:



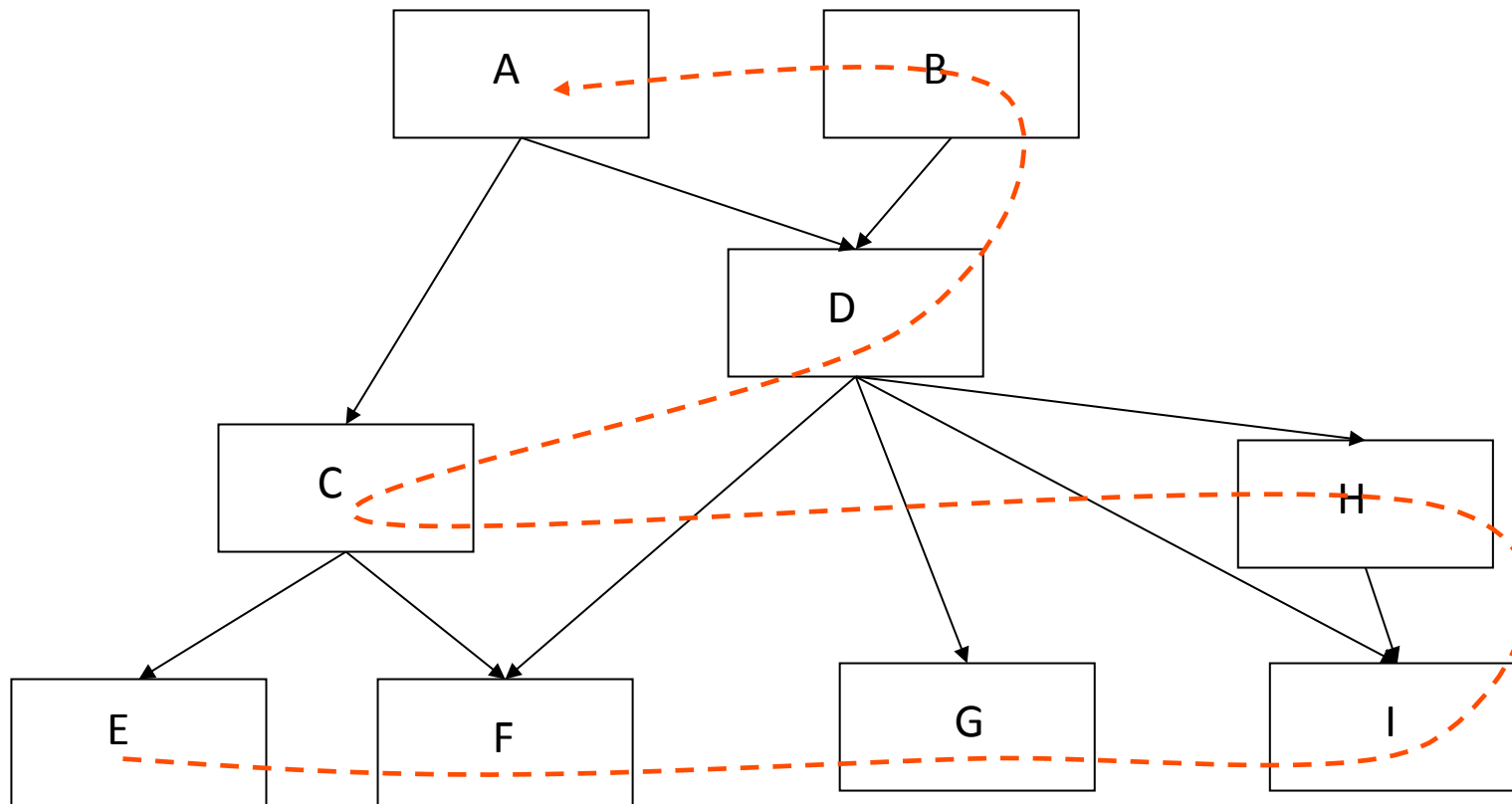
Top-down integration testing:



Bottom-Up Integration (1)

- ✧ Only terminal modules (i.e., the modules that do not call other modules) are tested in isolation
- ✧ Modules at lower levels are tested using the previously tested higher level modules
- ✧ Non-terminal modules are not tested in isolation
- ✧ Requires a module **driver** for each module to feed the test case input to the interface of the module being tested
 - However, stubs are not needed since we are starting with the terminal modules and use already tested modules when testing modules in the lower levels.

Bottom-up Integration (2)



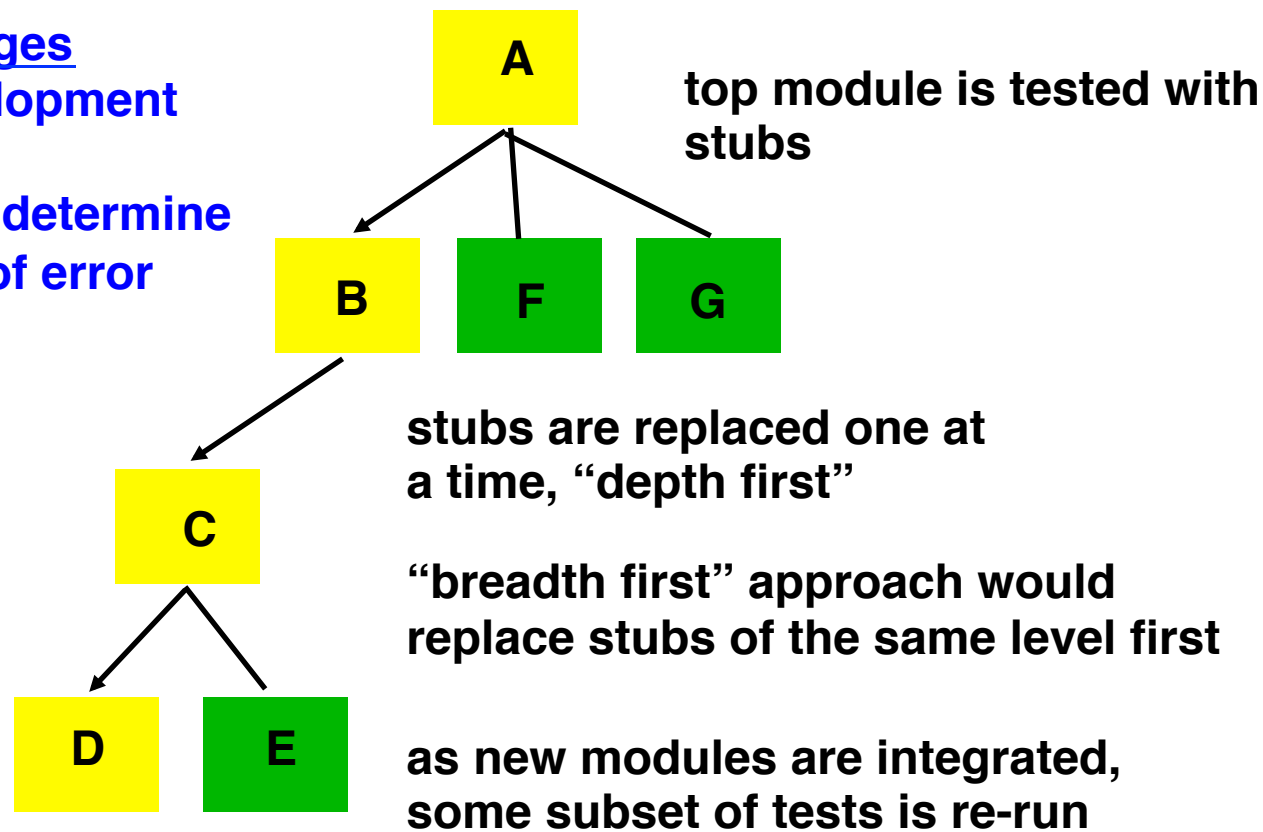
Top-down Integration (1)

- ✧ Only modules tested in isolation are the modules which are at the highest level
- ✧ After a module is tested, the modules directly called by that module are merged with the already tested module and the combination is tested
- ✧ Requires **stub** modules to simulate the functions of the missing modules that may be called
 - However, drivers are not needed since we are starting with the modules which is not used by any other module and use already tested modules when testing modules in the higher levels

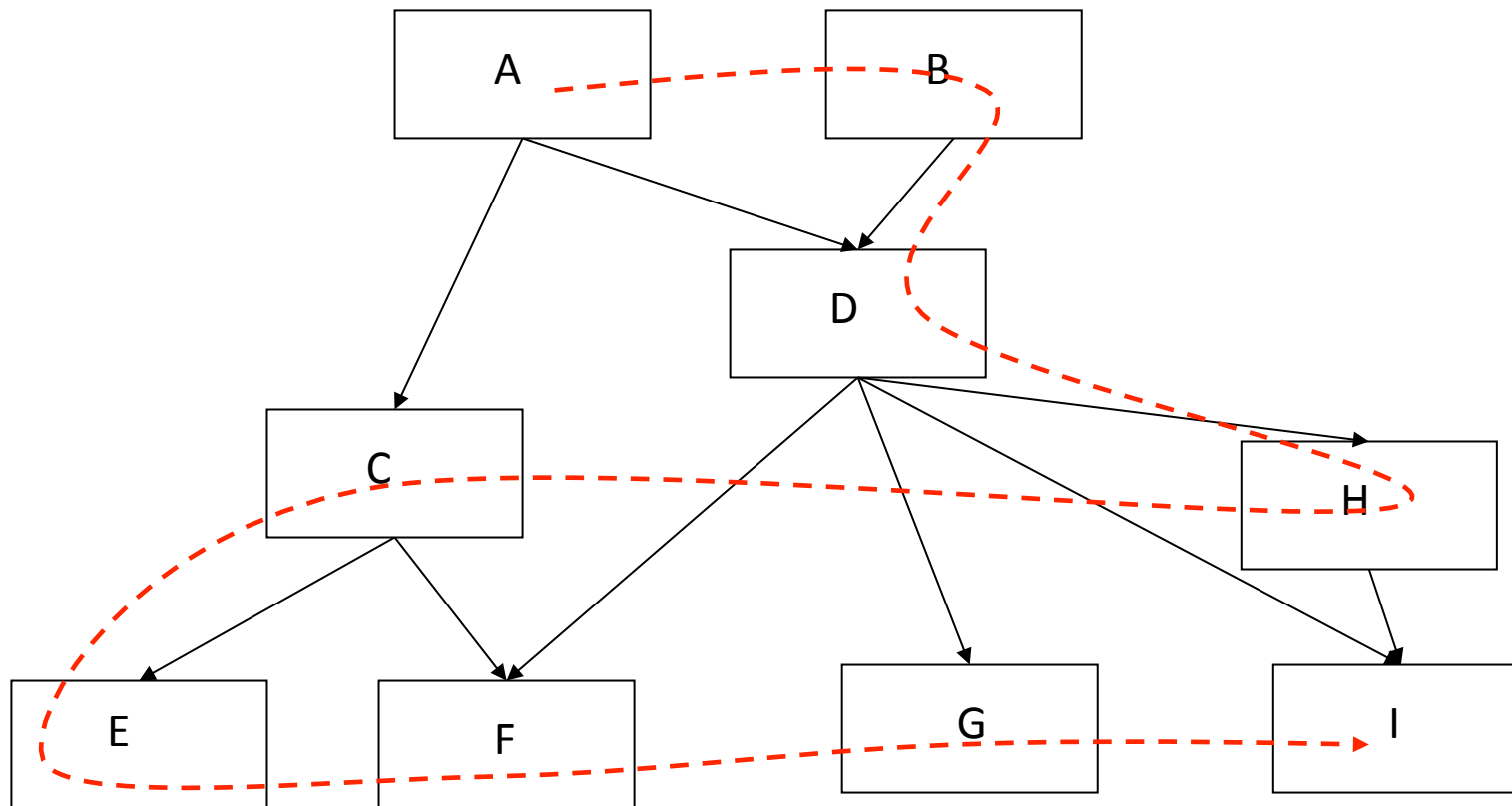
Top Down Integration (2)

Disadvantages

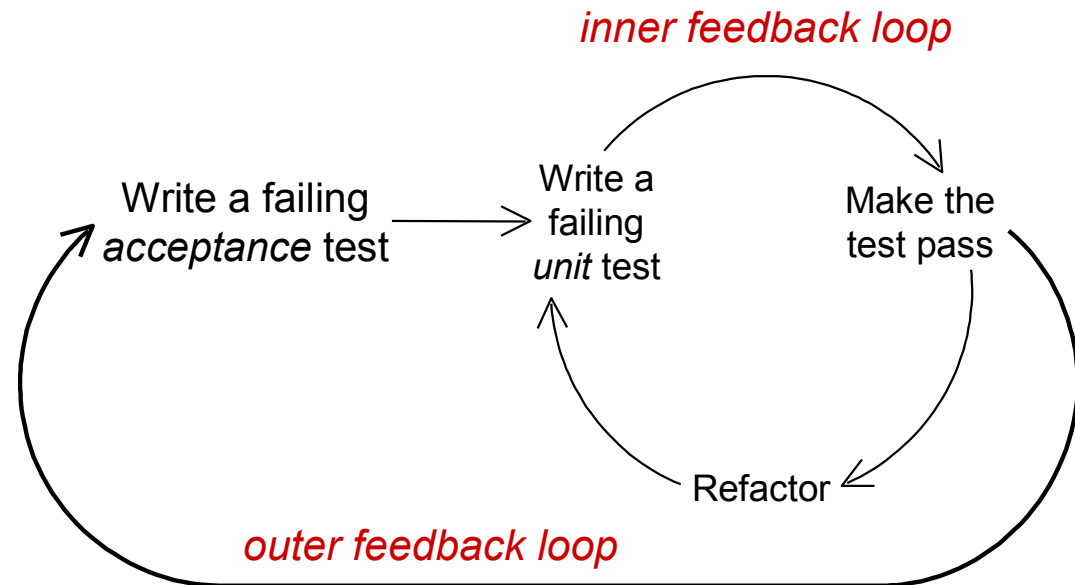
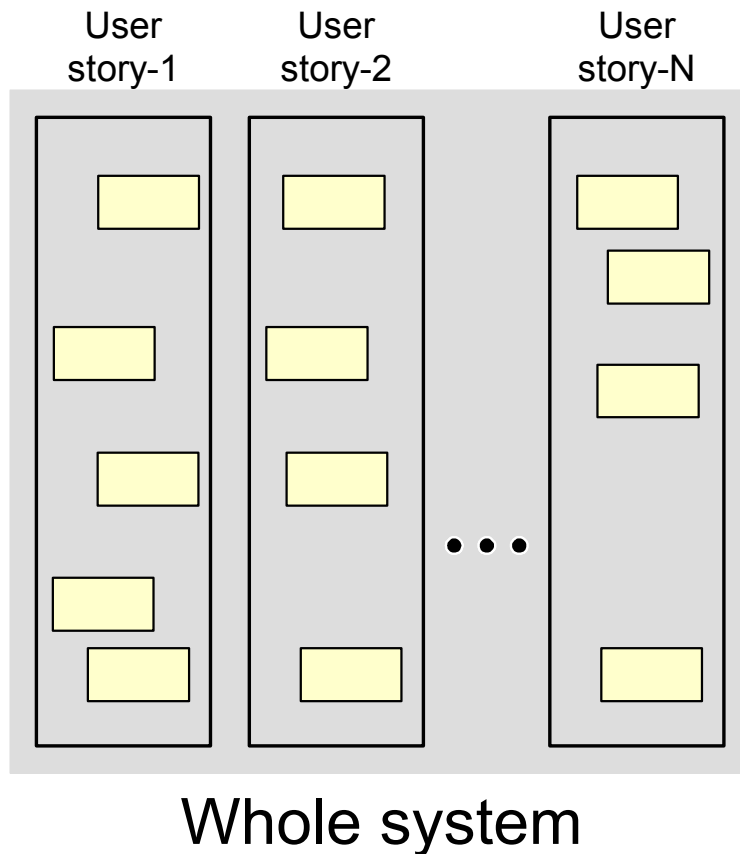
- Incur development overhead
- Difficult to determine real cause of error



Top-down Integration (3)



Vertical Integration Testing



Developing user stories:

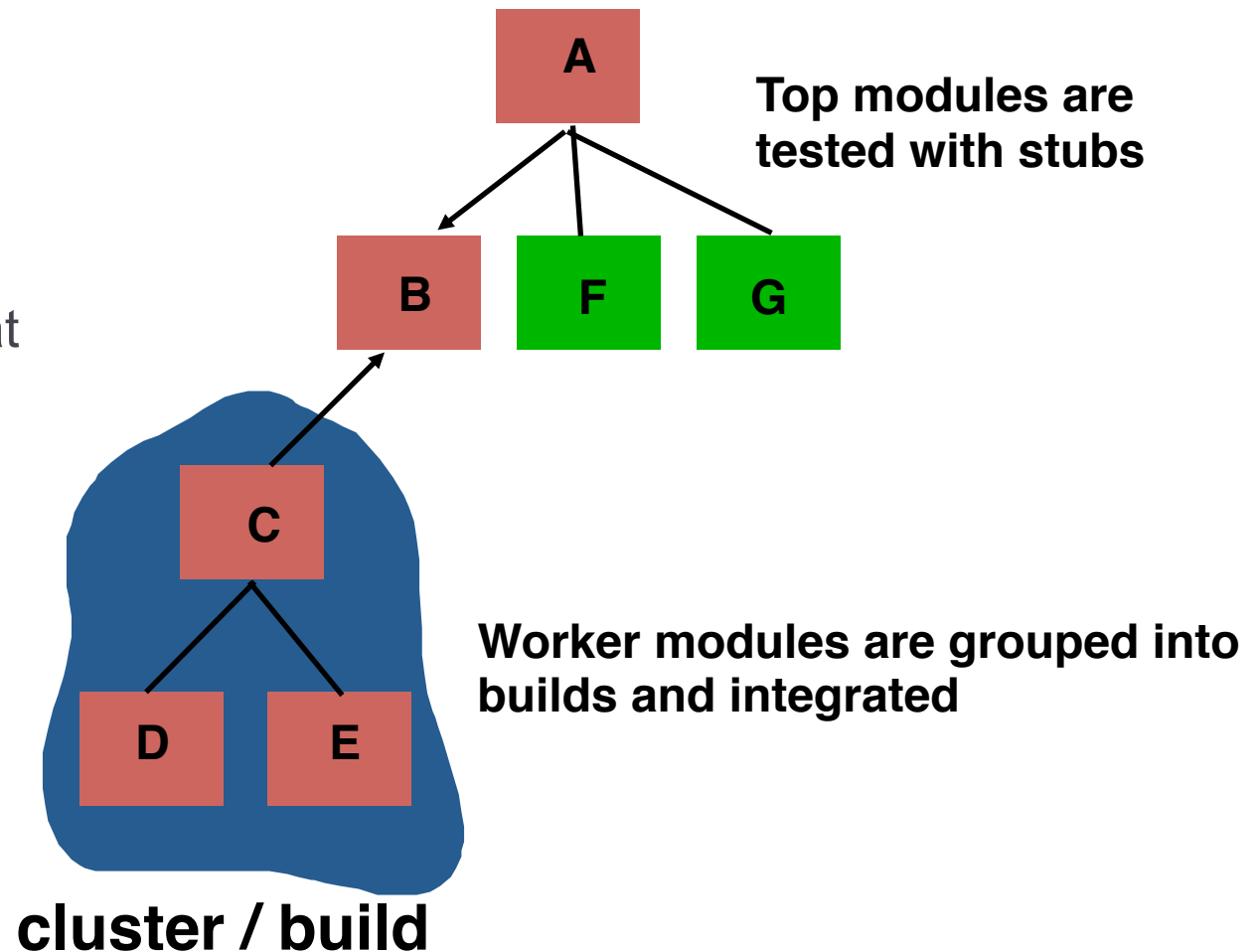
Each story is developed in a cycle that integrates

unit tests in the **inner feedback loop** and the

acceptance test in the **outer feedback loop**

Sandwich Integration

- Compromise between bottom-up and top-down testing
- Simultaneously begin bottom-up and top-down testing and meet at a predetermined point in the middle



Big Bang Integration

- Every module is unit tested in isolation
- After all of the modules are tested they are all integrated together at once and tested
- No driver or stub is needed
- However, in this approach, it may be hard to isolate the bugs

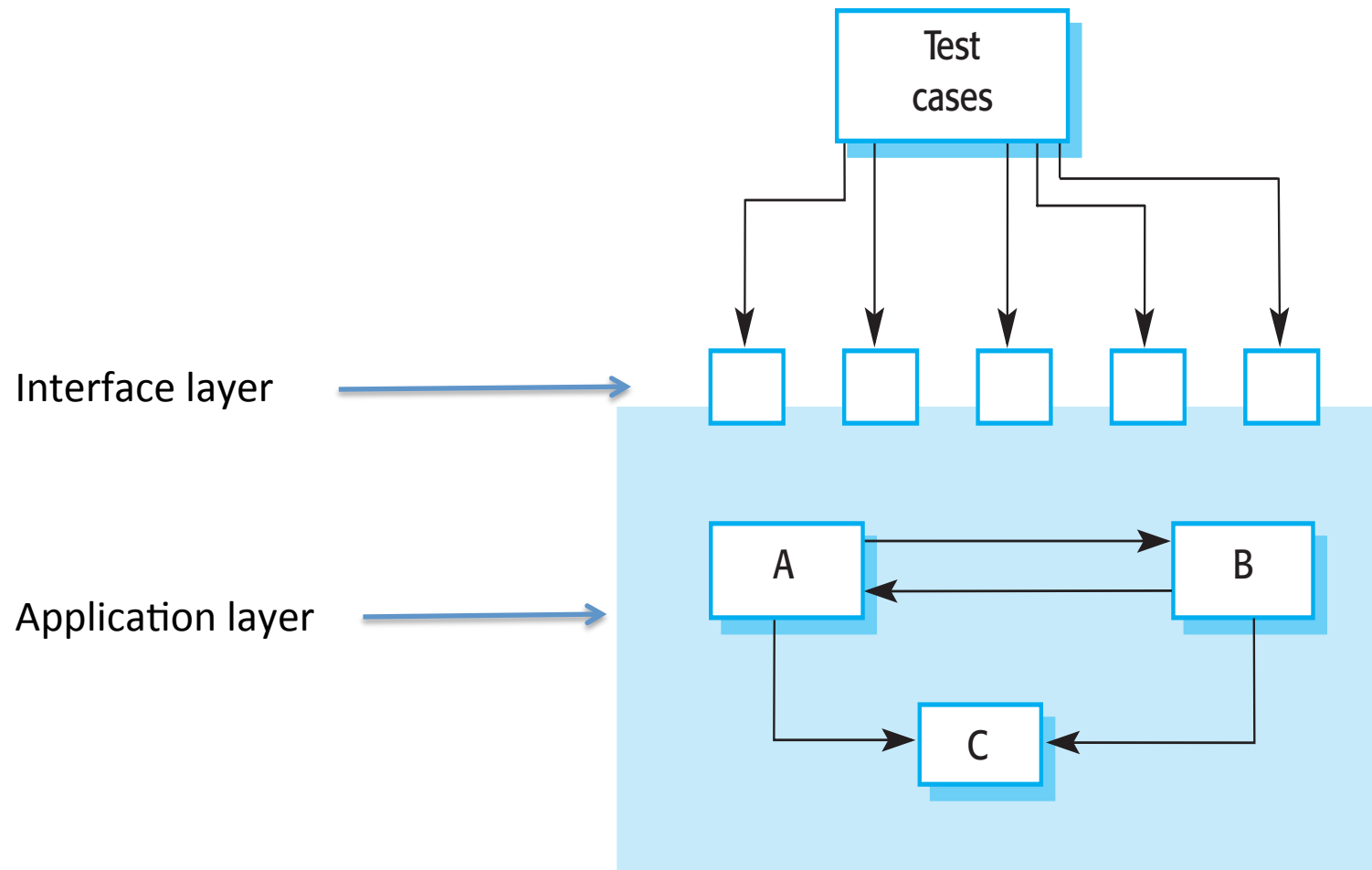
System Testing and Acceptance Testing

- ✧ System and Acceptance testing follows the integration phase
 - testing the system as a whole
- ✧ Test cases can be constructed based on the requirements specifications
 - main purpose is to assure that the system meets its requirements
- ✧ Manual testing
 - Somebody uses the software on a bunch of scenarios and records the results
 - Use cases and use case scenarios in the requirements specification would be very helpful here
 - manual testing is sometimes unavoidable: usability testing

System Testing and Acceptance Testing

- ✧ **Alpha testing** is performed within the development organization
- ✧ **Beta testing** is performed by a select group of friendly customers
- ✧ **Stress testing**
 - push system to extreme situations and see if it fails
 - large number of data, high input rate, low input rate, etc.

Interface Testing



Release Testing

- ✧ **Release testing** is the process of testing a particular release of a system that is intended for use outside of the development team.
- ✧ The primary goal of the release testing process is to convince the client of the system that it is good enough for use.
 - Release testing, therefore, has to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- ✧ Release testing is usually a black-box testing process where tests are only derived from the system specification.

User Testing

✧ Alpha testing

- Users of the software work with the development team to test the software at the developer's site.

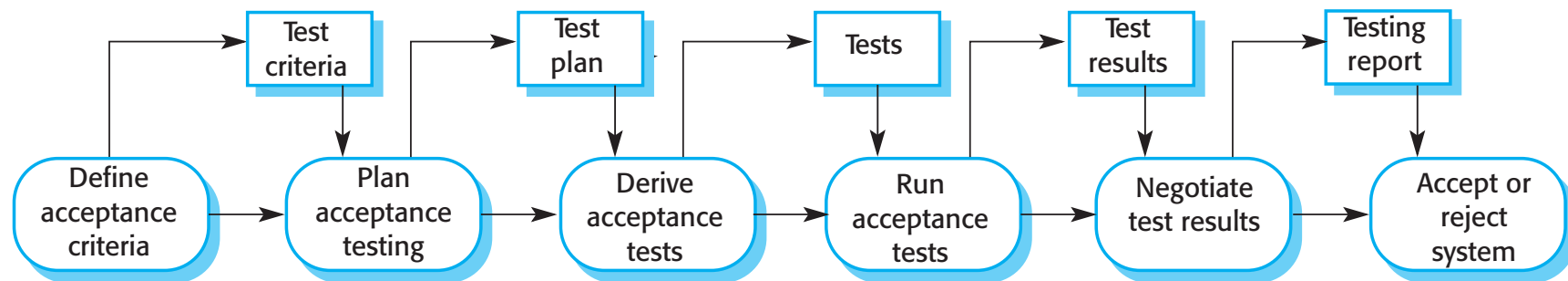
✧ Beta testing

- A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.

✧ Acceptance testing

- Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

The Acceptance Testing Process



Reference

- Your text book by Pressman (Chapter 17)
- Everett and Raymond: Software Testing. Wilkey and IEEE CS Press
- Black: Pragmatic Software Testing. Wiley.
- Perry: Effectice Methods of Software Testing, Wiley.
- Lee, and Yannakakis, “Principles and Methods of Testing Finite State Machines: A Survey”, Proceedings of The IEEE, Vol. 84, No. 8, August 1996,
- Chow, “Testing Software Design Modeled by Finite-State Machines” IEEE Transactions on Software Engineering, vol.4, no. 3, pp. 178-187, May 1978