Software Testing

L04. Test Classification

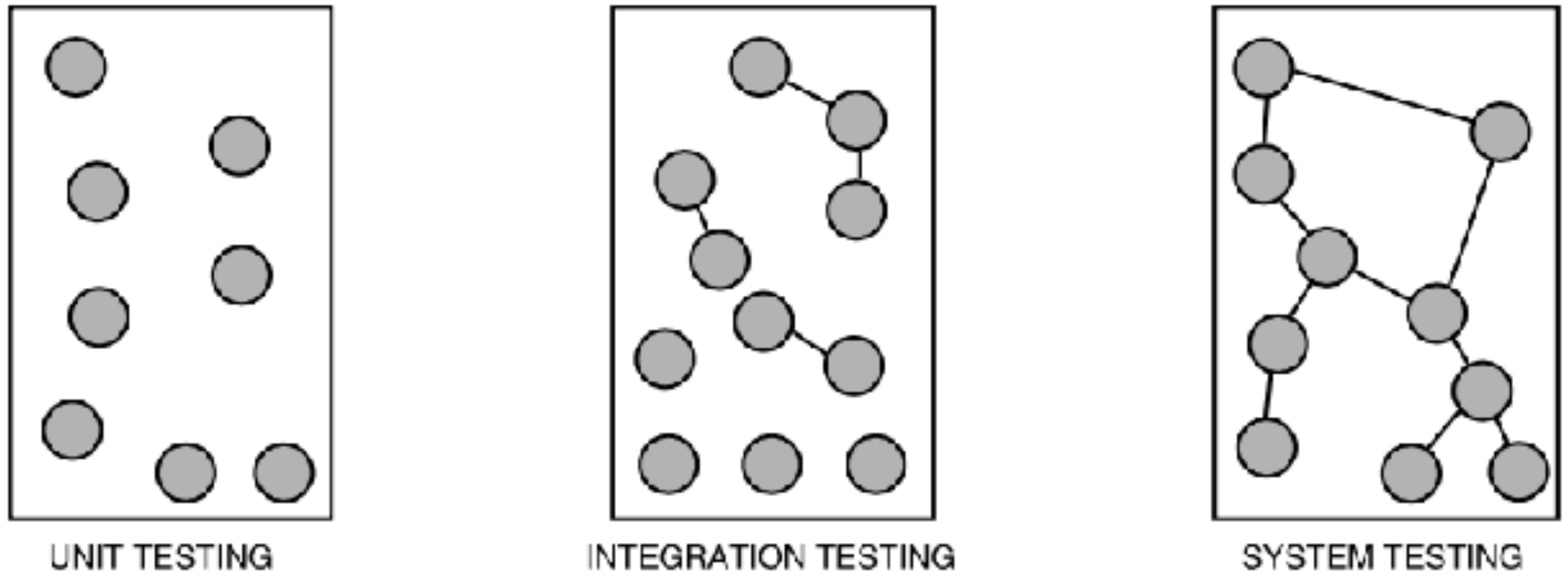
Christian Millán

Autumn 2020

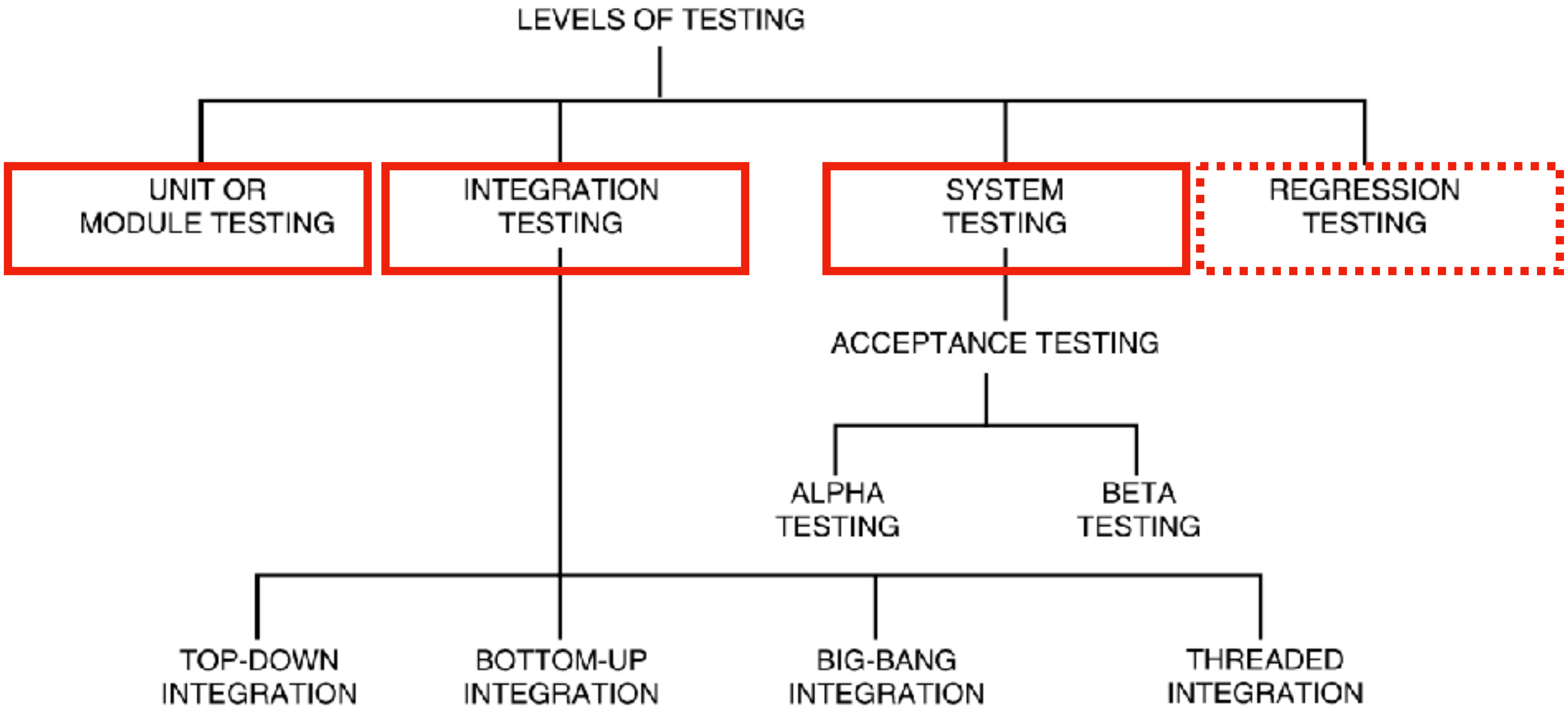
4. Test Classification

4.1. Level test

1. **Unit test**
2. **Integration test**
3. **System test**



4.1 Level Test



4.1.1. Unit Testing

It is the process of **taking a** module (an **atomic unit**) and **running it in isolation** from the rest of the software product by using prepared **test cases** and **comparing** the **actual results** with the **results predicted** by the specification and design module.”

- It is a **white-box** testing technique.

4.1.1. Unit Testing

Importante of the unit test

1. Because modules are being tested individually, testing becomes easier.

2. It is more exhaustive.

3. Interface errors are eliminated.

4.1.2. Integration Testing

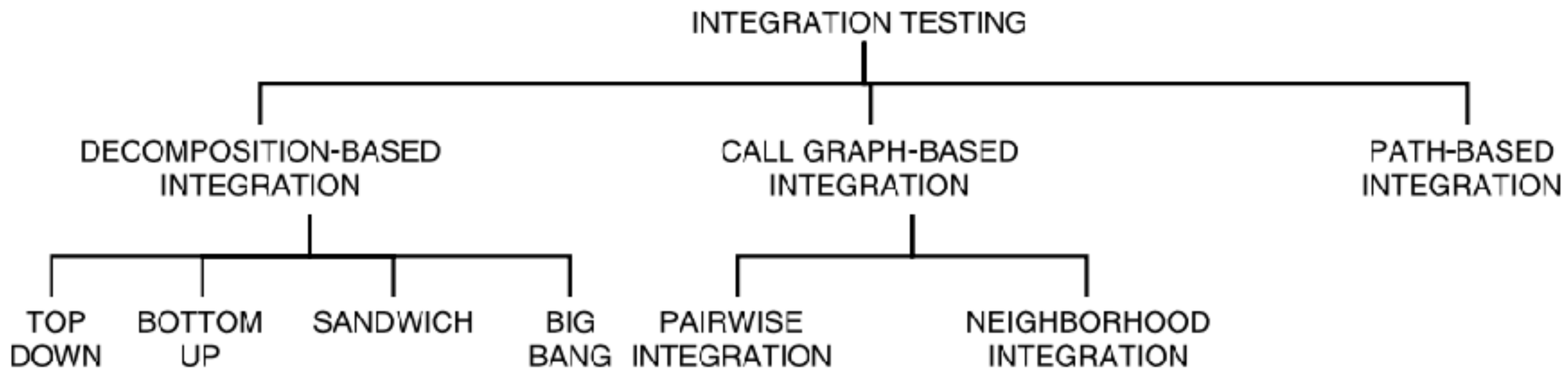
Type of testing and a **phase** of testing.

4.1.2. Integration Testing

- Integration is defined as the set of interactions among components.
- Testing the interaction between the **modules** and interaction with **other systems externally** is called **integration testing.**

4.1.2. Integration Testing

- Testing the interaction between the **modules** and interaction with **other systems externally** is called **integration testing.**



4.1.2. Integration Testing

4.1.2.1. Decomposition-Based Integration

When it is tested the **functional** decomposition of the system

The goal of decomposition-based integration is to test the interfaces among separately tested units.

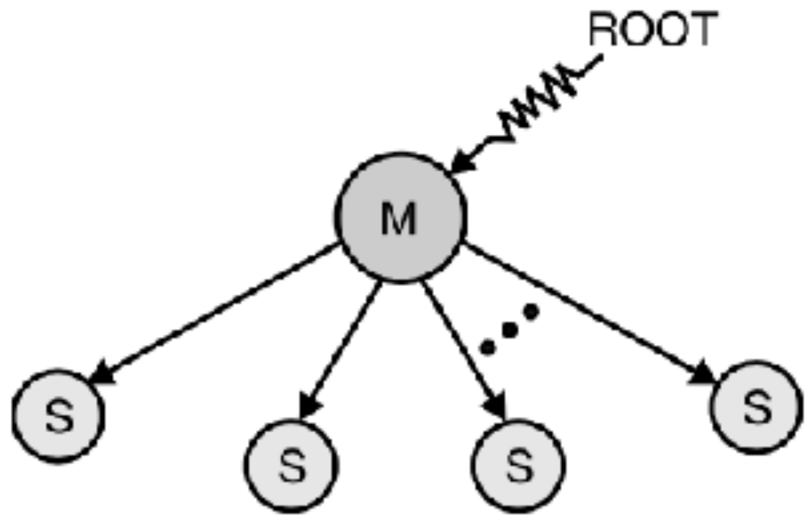
- Top-down
- Botton-up
- Sandwich
- Big bang

4.1.2. Integration Testing

4.1.2.1.1. Top-down

It begins with the **main program**, i.e., the root of the tree.

Any lower-level unit that is called by the main program appears as a “stub.”

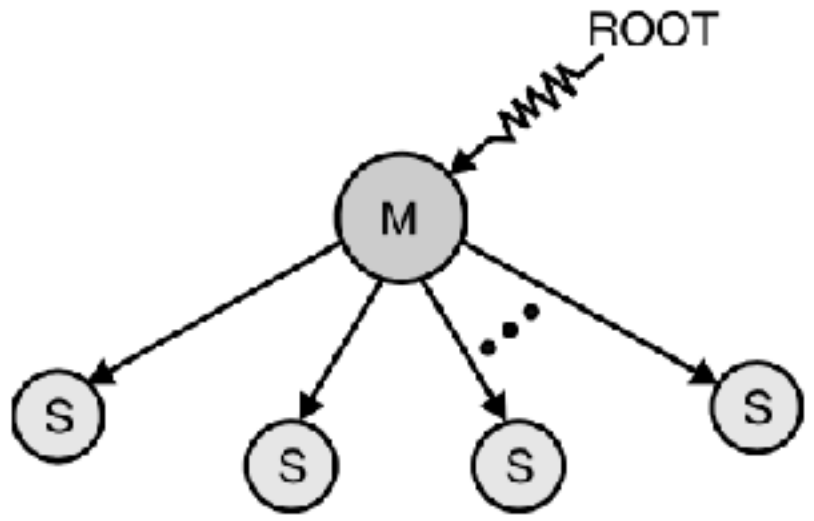


4.1.2. Integration Testing

4.1.2.1.1. Top-down

A **stub** is a piece of throw-away code that emulates a called unit.

When we are convinced that the main program logic is correct, we gradually **replace** the **stubs** with the **actual code**

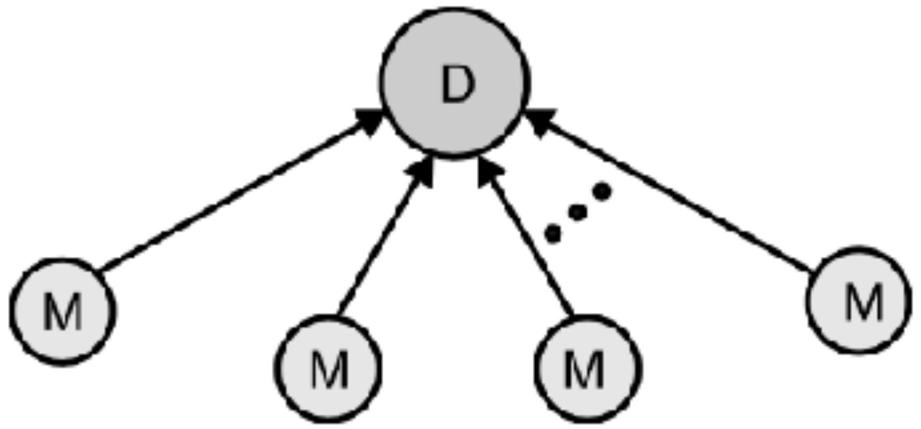


4.1.2. Integration Testing

4.1.2.1.2. Bottom-up

Driver modules that emulate units at the next level up in the tree instead of stubs

In bottom-up integration, we start with the leaves of the decomposition tree and test them with specially coded drivers.



4.1.2. Integration Testing

4.1.2.1.3. Sandwich

It is a combination of top-down and bottom-up integration.

It is performed initially with the use of stubs and drivers.

A **driver** is a function which redirects the request to some other component and **stubs** simulate the behavior of a missing component.

4.1.2. Integration Testing

4.1.2.1.4. Big-bang

Instead of integrating component by component and testing, this approach waits until all the components arrive and one round of integration testing is done.

It reduces testing effort and removes duplication in testing for the multi-step component integrations.

4.1.2. Integration Testing

4.1.2.2. Call Graph-Based Integration

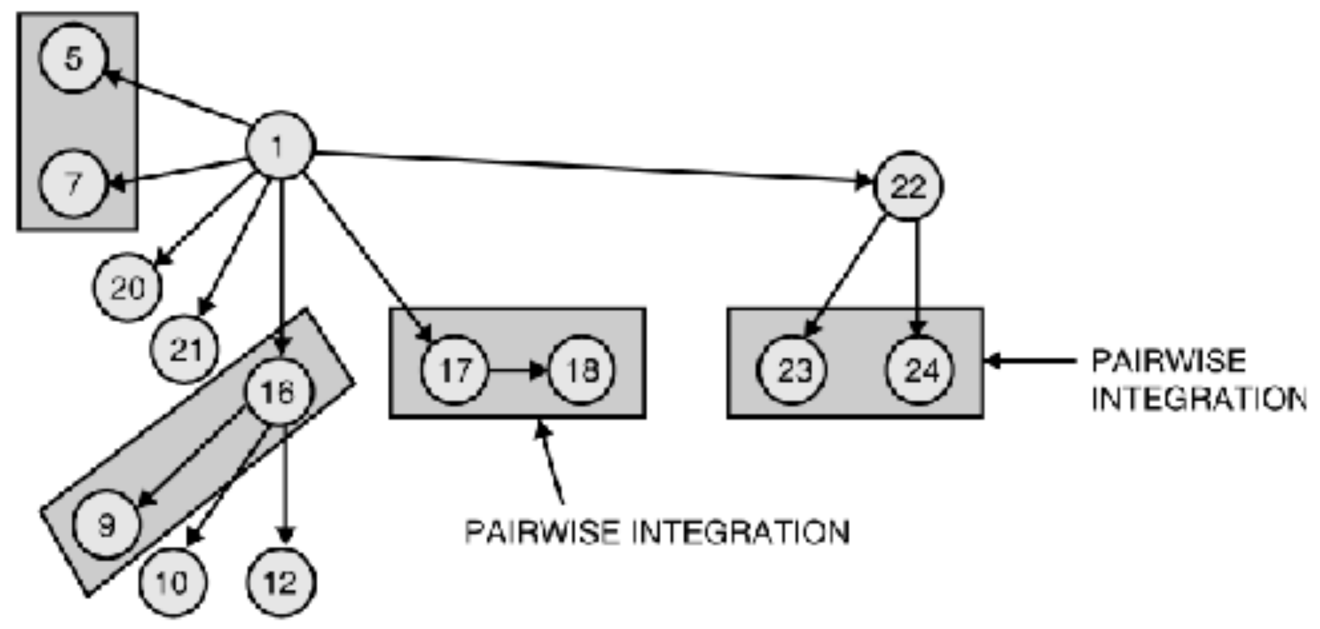
The call graph-based technique is focused at **structural** testing.

- Pairwise Integration
- Neighborhood Integration

4.1.2. Integration Testing

4.1.2.2.1. Pairwise Integration

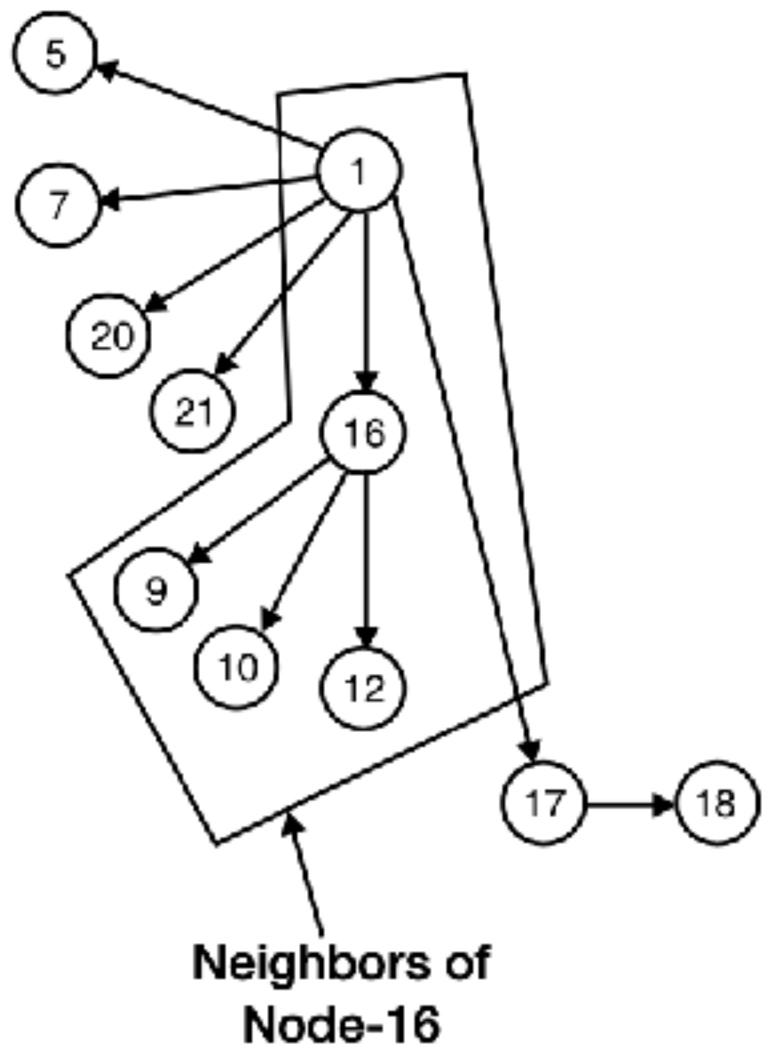
- Eliminate stubs and drivers
- It is a drastic reduction in stub/driver development.



4.1.2. Integration Testing

4.1.2.2.2. Neighborhood Integration

- The neighborhood of a node in a graph is the set of nodes that are one edge away from the given node
- The end result is that the neighborhoods are essentially the sandwiches.



4.1.2. Integration Testing

4.1.2.3. Path-Based Integration

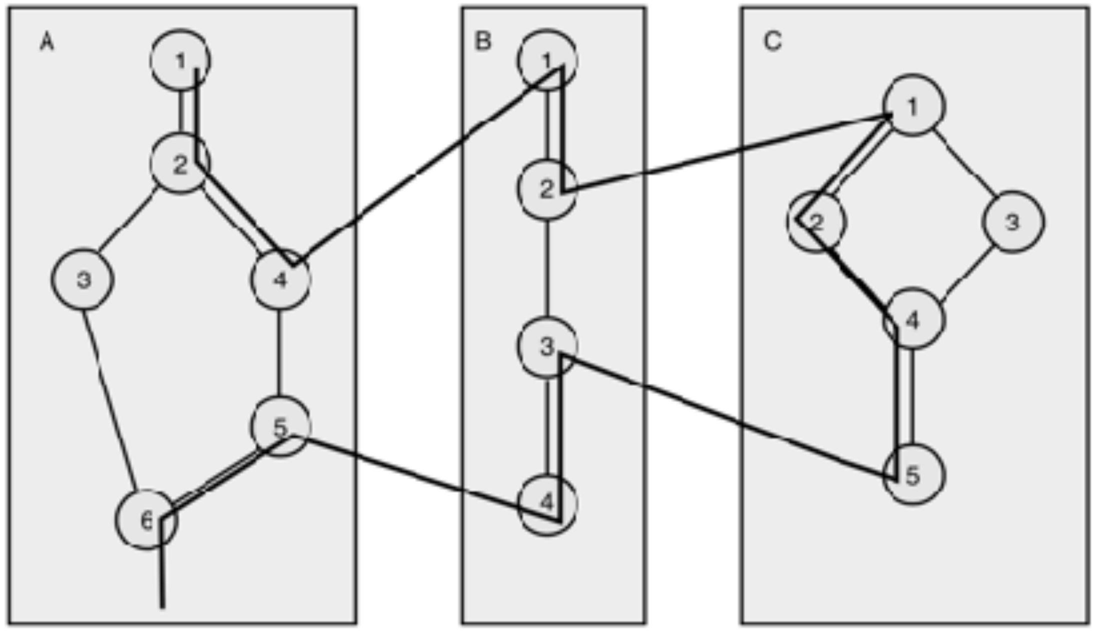
"Instead of testing interfaces among separately developed and tested units, we focus on interactions among these units."

4.1.2. Integration Testing

4.1.2.3. Path-Based Integration

Terminology

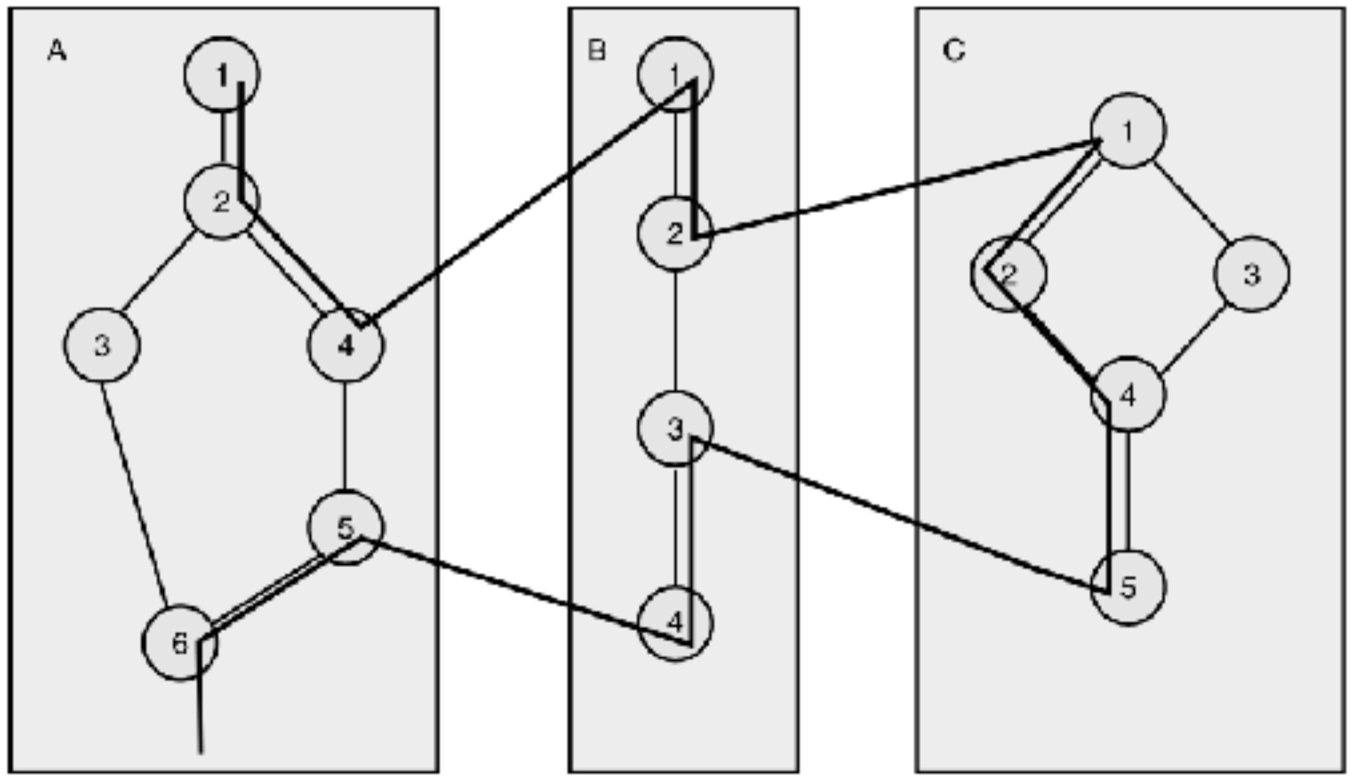
- Statement fragment
- Source node
- Sink node
- Module execution path (MEP)
- Massage
- Module-to-module path (MM-path)
- Module-to-module path graph (MM-path graph)



4.1.2. Integration Testing

4.1.2.3. Path-Based Integration

- Module-to-module path (MM-path) describes sequences of module execution paths that include transfers of control among separate units.

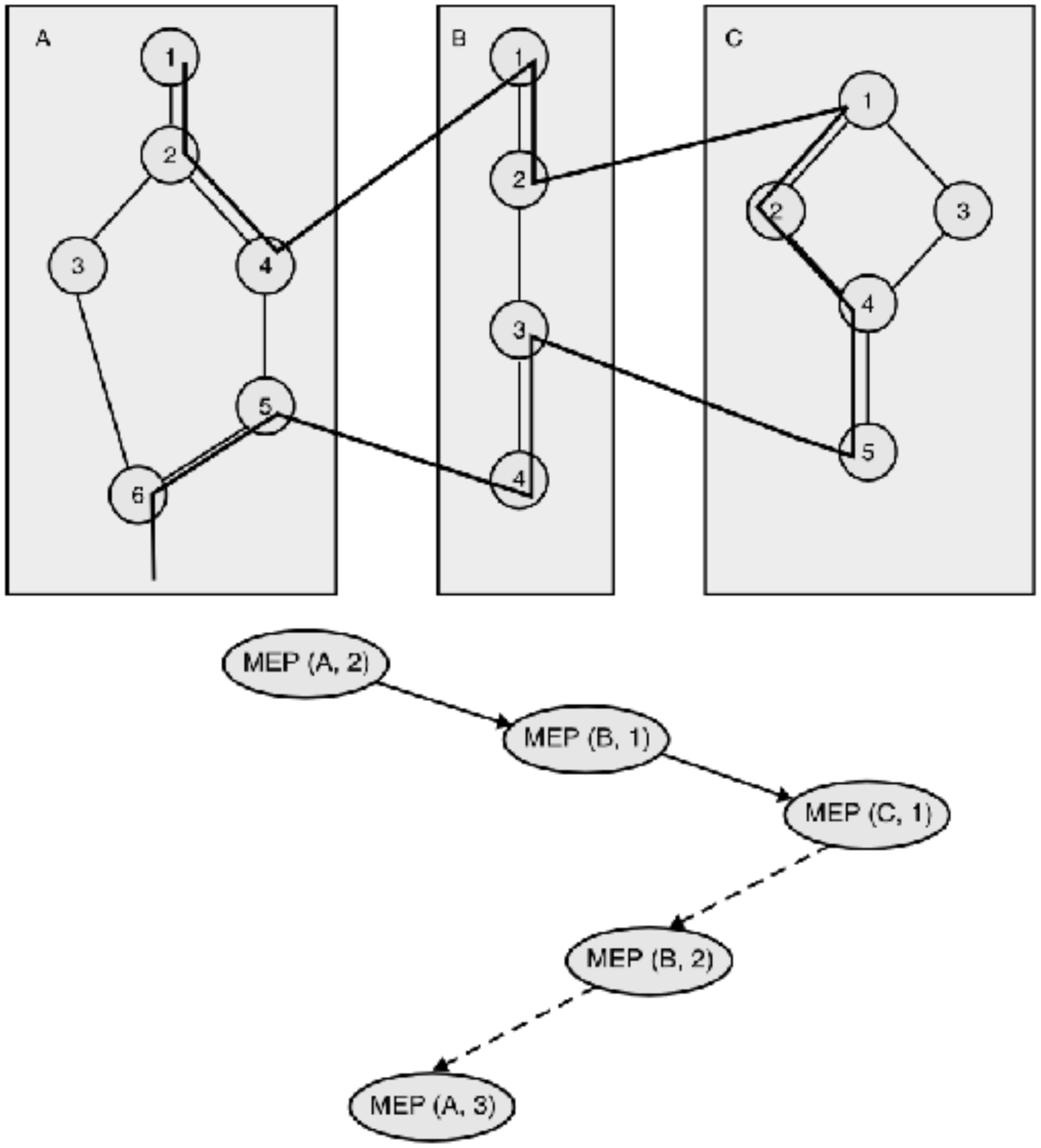


4.1.2. Integration Testing

4.1.2.3. Path-Based Integration

Module	Source	Sink
A	1, 5	4, 6
B	1, 3	2, 4
C	1	5

- MEP(A,1) = <1, 2, 3, 6>
- MEP(A,2) = <1, 2, 4>
- MEP(A,3) = <5, 6>
- MEP(B,1) = <1, 2>
- MEP(B,2) = <3, 4>
- MEP(C,1) = <1, 2, 4, 5>
- MEP(C,2) = <1, 3, 4, 5>



4.1.3. System Testing

System testing focuses on a complete, integrated system to evaluate compliance with specified requirements.

Tests are made on characteristics that are only present when the entire system is run.

4.1.3. System Testing

The testing that is conducted on the complete integrated products and solutions to evaluate system compliance with **specified requirements** on **functional** and **non functional** aspects is called **system testing**.

4.1.3. System Testing

It is done after unit, component, and integration testing phases.

System testing is the only testing phase that tests both functional and non functional aspects of the product.

4.1.3. System Testing

System testing is done to:

1. Provide independent perspective in testing as the team becomes more quality centric.
2. Bring in customer perspective in testing.
3. Provide a “fresh pair of eyes” to discover defects not found earlier by testing.
4. Test product behavior in a holistic, complete, and realistic environment.

4.1.3. System Testing

5. Test both functional and non functional aspects of the product.
6. Build confidence in the product.
7. Analyze and reduce the risk of releasing the product.
8. Ensure all requirements are met and ready the product for acceptance testing.

4.1.3. System Testing

System testing = Functional testing + Non functional testing

Functional testing	Non functional testing
It involves the product's functionality.	It involves the product's quality factors.
Failures, here, occur due to code.	Failures occur due to either architecture, design, or due to code.
It is done during unit, component, integration, and system testing phase.	It is done in our system testing phase.
To do this type of testing only domain of the product is required.	To do this type of testing, we need domain, design, architecture, and product's knowledge.
Configuration remains same for a test suite.	Test configuration is different for each test suite.

4.1.3. System Testing

Functional testing helps in verifying what the system is supposed to do. It has only two results—requirements met or not met.

Non functional testing is performed to verify the quality factors such as reliability, scalability, etc. It requires the expected results to be documented in qualitative and quantifiable terms.

4.2. Type of Tests

- Acceptance testing
- Alpha-Beta testing
- Regression Testing

4.2.1. Acceptance Testing

The goal of acceptance testing is to establish confidence in the system, part of the system or specific non-functional characteristics, e.g. usability of the system.

Acceptance testing is not focused in find defects. It is focused in acceptance testing.

4.2.1. Acceptance Testing

Types of acceptance testing:

- User acceptance test (functionality)
- Operational acceptance test (requirements for operation)
- Contract acceptance test
- Compliance acceptance test or regulation acceptance test

4.2.3. Alpha-Beta testing

If the system has been developed for the mass market, the testing of the individual users or customers is not practical or even possible in some cases.

Acceptance test is divided into two stages:

- Alpha testing
- Beta testing

4.2.3. Alpha-Beta testing

Alpha testing

- Takes place in the developer site
- Potential users
- Developers observe the users to detect problems

Beta testing

- Send to potential users
- Real-world conditions
- Users send records of incidents

4.2.5. Regression testing

Once deployed, a system is often corrected, changed or extended. Tests in this phase is called **maintenance test**.

Maintenance test consist in two parts:

- Testing the changes
- Regression test

4.2.5. Regression testing

Regression test shows that the rest of the system has not been affected by the maintenance work.