fit@hcmus

Software Testing

CSC13003

Software Testing Life Cycle

Content

- Software testing life cycle
- 7 principles of software testing
- Levels of software testing
- Types of software testing

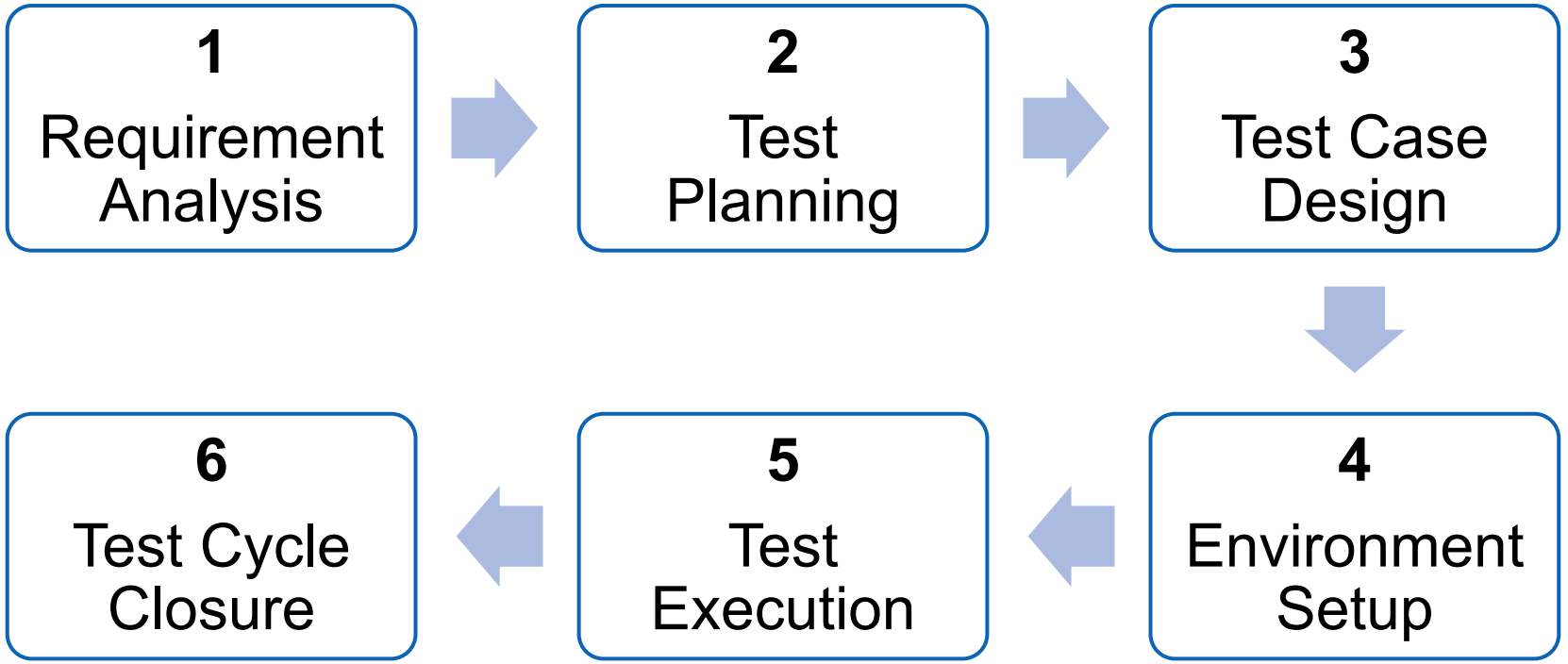
Content

- **Software testing life cycle**
- 7 principles of software testing
- Levels of software testing
- Types of software testing

What is STLC?

The software testing life cycle is the process of executing different activities during testing

Phases of STLC



1. Requirement analysis

- Analyze requirements to understand scope of testing
- May consult with stakeholders to clarify requirements
- Identify any potential challenges or risks
- Create Requirements Traceability Matrix
- Evaluate the ability to automate testing

1. Requirement analysis

- Entry criteria
  - Requirement Specification
  - Application Architecture
- Deliverables
  - Requirements Traceability Matrix
  - Automation Feasibility Report

2. Test planning

- Define test objectives and scope
- Develop a test strategy including test methods
- Identify the test environment and resources
- Estimate the time and cost required for testing
- Identify the test deliverables and milestones
- Assign test team roles and responsibilities

2. Test planning

- Entry criteria
  - Project Plan
  - Acceptance Criteria
  - Requirement Specification
- Deliverables
  - Test Plan

3. Test case design

- Design test cases and generate test data
- Create automation scripts, if appliable
- Update the Requirements Traceability Matrix
- Test cases
  - Inputs
  - Test steps
  - Expected result

3. Test case design

- Entry criteria
  - Test Plan
  - Requirement Specification
- Deliverables
  - Test Cases
  - Test Data
  - Automation Scripts

4. Environment setup

- Prepare hardware and software list
- Configure and deploy test environments
- Perform smoke tests to verify readiness

4. Environment setup

- Entry criteria
  - Test Plan
  - Test Cases
  - System Architecture
- Deliverables
  - Fully functional test environment
  - Smoke Test Results

5. Test execution

- Run test cases in the deployed environment
- Collect and analyze test results
- Compare actual results with expected results
- Report defects found during test execution

5. Test execution

- Entry criteria
  - Test Cases
  - Test Data
  - Automation Script
- Deliverables
  - Test Results
  - Defect Reports

6. Test cycle closure

- Ensure that all testing activities have been completed
- Document the testing process and any lessons learned
- Prepare Test Summary Report
- Prepare Test Closure Report

6. Test cycle closure

- Entry criteria
  - Test Cases
  - Test Results
  - Defect Reports
- Deliverables
  - Test Summary Report
  - Test Closure Report

Content

- Software testing life cycle
- **7 principles of software testing**
- Levels of software testing
- Types of software testing

7 principles of software testing

- #1: Testing shows the presence of defects
  - Don’t talk about the absence of defects
  - Reduces the probability of undiscovered defects
- #2: Exhaustive testing is not possible
  - Need the optimal amount of testing
  - Based on the risk assessment
- #3: Early testing
  - Testing should start as early as possible
  - Much cheaper to fix a defect in the early stages of testing

7 principles of software testing

- #4: Defect clustering
  - A small number of modules contain most of the defects detected
  - Pareto Principle: 80% of the defects are found in 20% of the modules
- #5: Pesticide paradox
  - If the same set of repetitive tests are conducted, the method will be useless for discovering new defects
  - The test cases need to be regularly reviewed & revised, adding new & different test cases

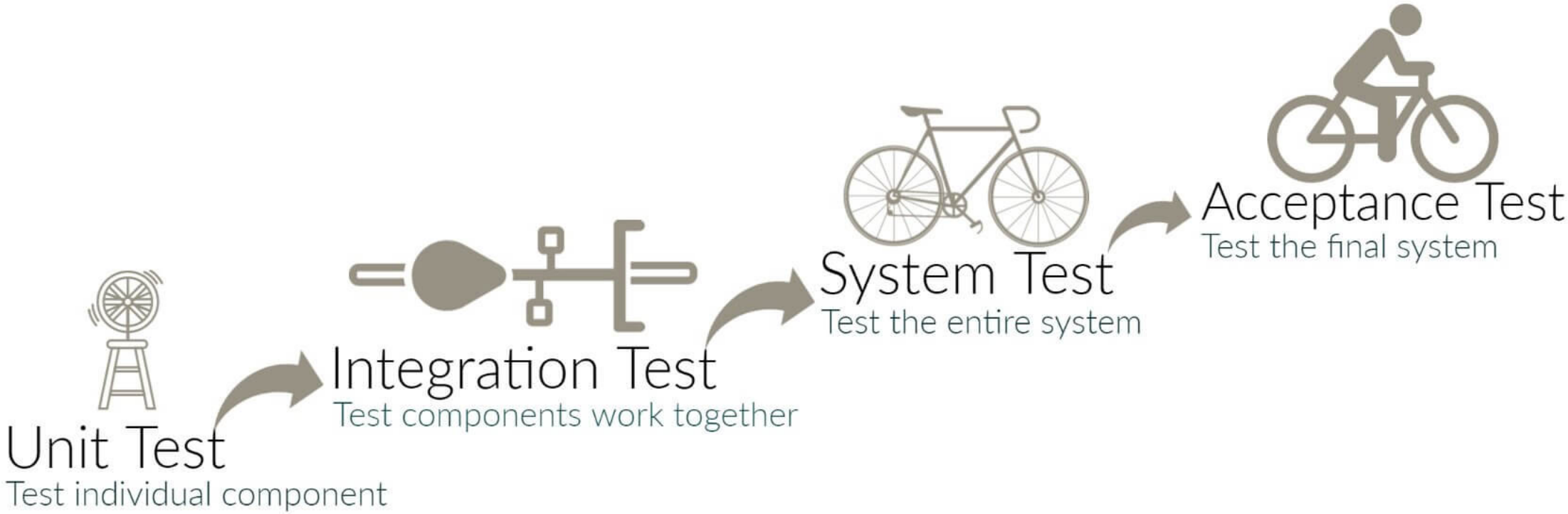
7 principles of software testing

- #6: Testing is context dependent
  - The way you test an e-commerce site will be different from the way you test a commercial off the shelf application
  - Use a different approach, methodologies, techniques, and types of testing depending upon the application type
- #7: Absence of error – fallacy
  - It is possible that software which is 99% bug-free is still unusable
  - Software testing is not mere finding defects, but also to check that software addresses the business needs

Content

- Software testing life cycle
- 7 principles of software testing
- **Levels of software testing**
- Types of software testing

Four levels of testing

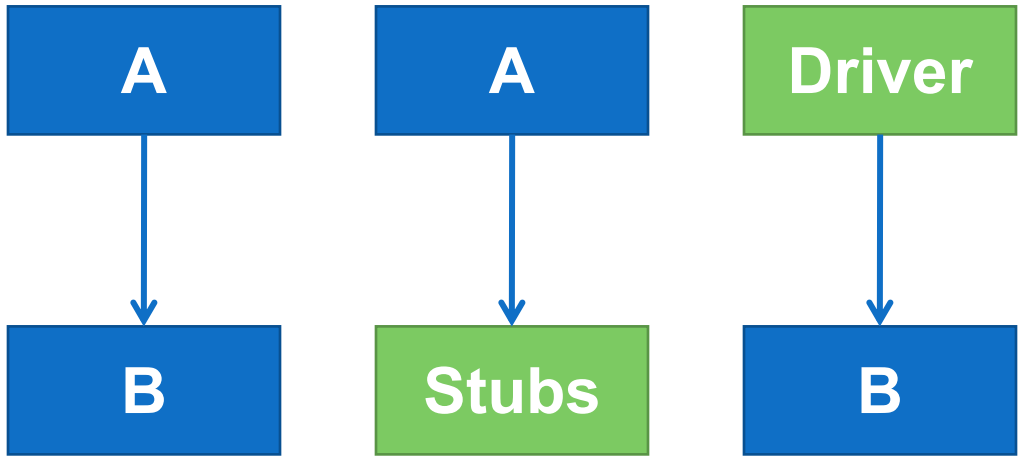


Unit testing

- Component/Module/Program testing
- Each unit is independently tested
- Performed by developers
- Bugs are fixed immediately, no need to report

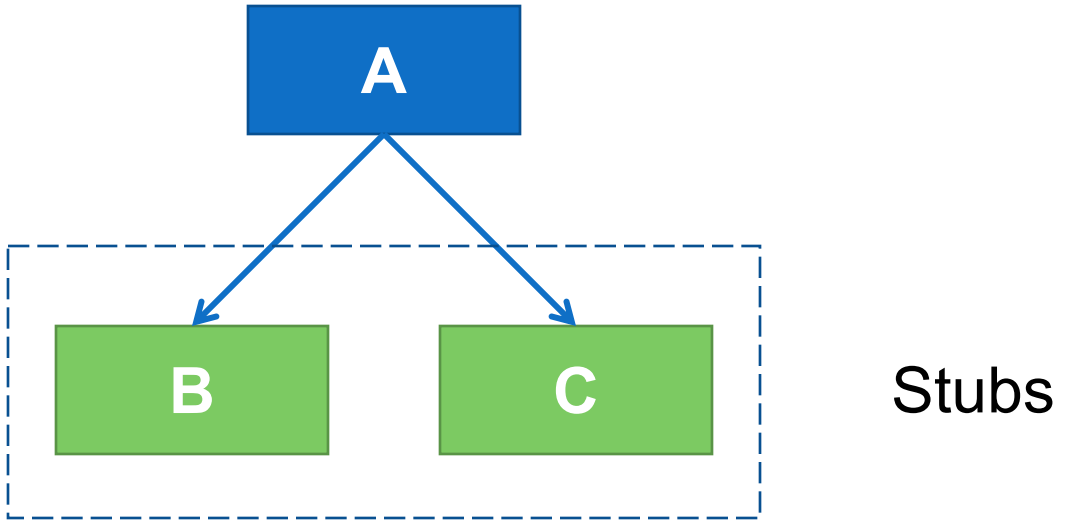
Unit testing

- Stubs and Driver are simulator units
- Return value
  - Static value
  - Input value



Unit testing

- Unit A calls Units B and C
- Test Unit A independently
  - Replace Units B and C with simulator units (Stubs)



Unit testing

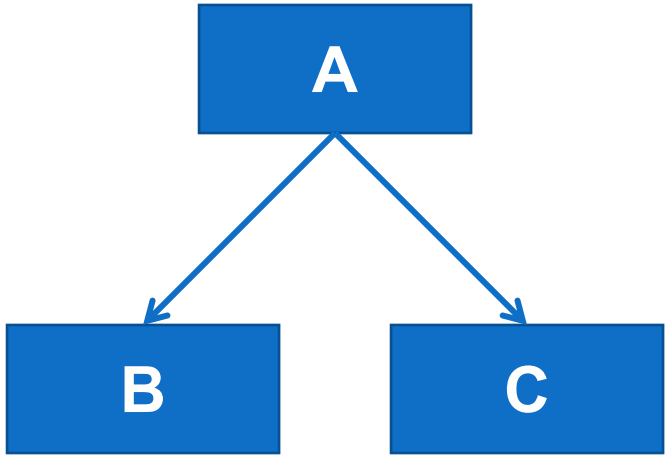
- Testing tools
  - Write source code directly
  - Unit testing framework
  - Mocking framework
  - Dependency injection and IoC container

Integration testing

- Test two or more units/sub-systems
- Test the interface/interaction between units
- Performed by developer and tester
- Integration approaches
  - Big-bang integration
  - Incremental integration

Integration testing

- Big-bang integration
  - Integrate all units at once
  - Test them all as one unit
  - Advantages
    - Its suitability for testing small systems
    - Saving time and speeding up application deployment
  - Disadvantages
    - Locating the source of defects can be difficult
    - Could miss some interface links or bugs
    - Must wait until all units are available
    - High-risk critical units are not prioritized for testing

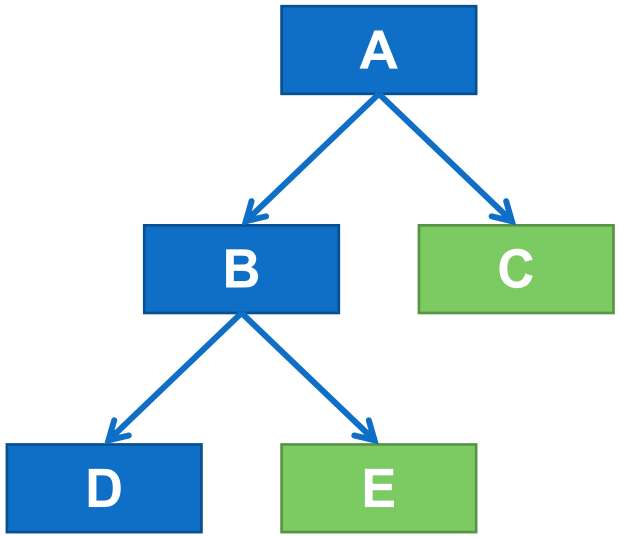


Integration testing

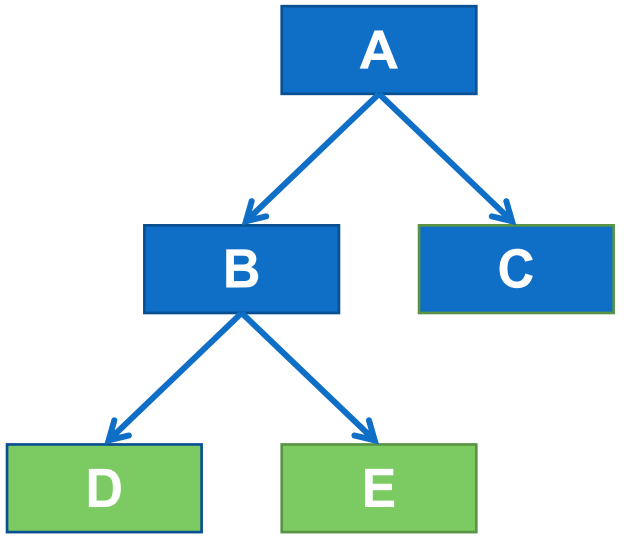
- Incremental integration
  - Start with one unit, add each unit incrementally, and test along a baseline
  - Advantage
    - Bugs are easy to find and fix
    - Can start early
  - Disadvantage
    - Difficult to simulate complex units
  - Approaches
    - Top-down
    - Bottom-up
    - Sandwich

Integration testing

- Top-down Integration
  - Test high-level units first, then gradually integrate lower-level units



Depth-first integration



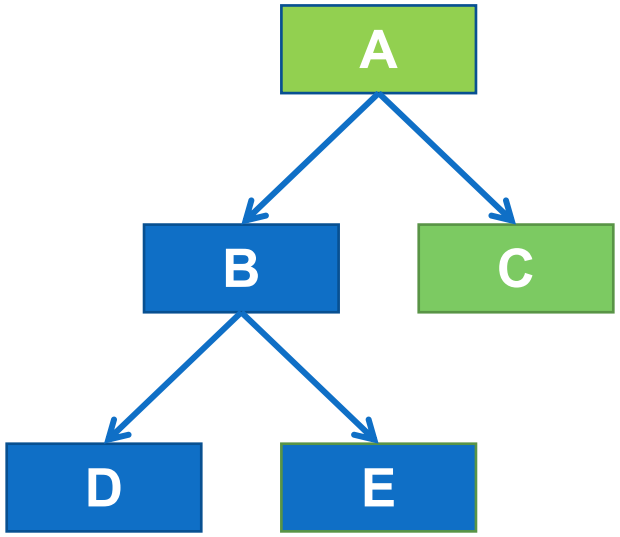
Breath-first integration

Integration testing

- Top-down Integration
  - Advantages
    - Easier to identify defects and isolate their sources
    - Check important units first, more likely to find critical design flaws
    - Possible to create an early prototype.
  - Disadvantages
    - The examination of lower-level modules can take a lot of time
    - When too many testing stubs are involved, the testing process can become complicated.

Integration testing

- Bottom-up Integration
  - The lowest level units are integrated into groups that represent a function of the software

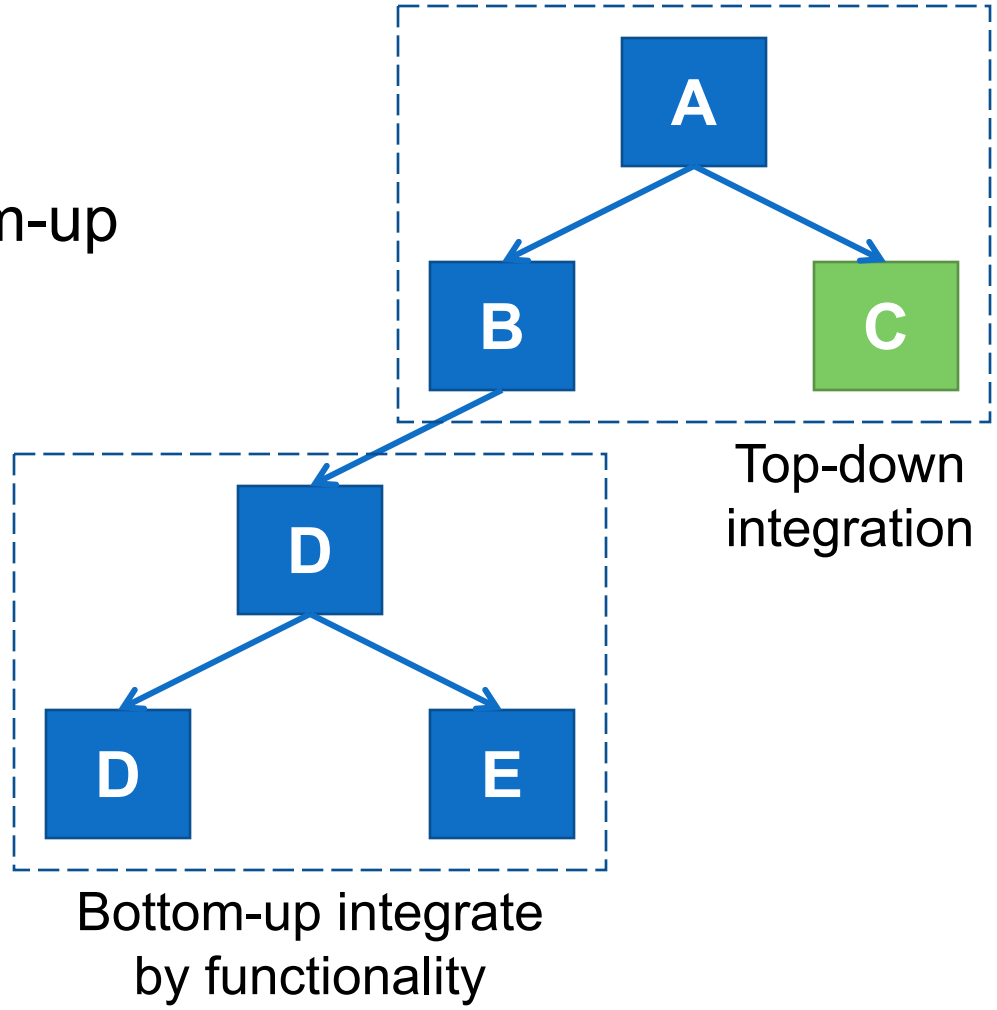


Integration testing

- Bottom-up Integration
  - Advantages
    - Easier to find and localize faults
    - Don't have to wait for all units to be available for testing
  - Disadvantages
    - Testing all units can take a lot of time
    - Critical units are tested only in the final stages
    - Testing can be complicated if consisting of multiple low-level units
    - Not possible to create an early prototype

Integration testing

- Sandwich testing
  - Combine Top-down and Bottom-up



System testing

- The final step of integration testing
- Test the complete and fully integrated system
- Perform by tester and business analyst
- Include both functional and non-functional testing

Acceptance testing

- Final step of validation
- Ensure that the system meets user expectations
- Performed by end-user

Acceptance testing

- Alpha testing and Beta testing
  - Similarity
    - When the software is stable
    - Get feedback on bugs, expectations, suggestions
  - Difference
    - Alpha testing is done in a development environment
    - Beta testing is done in a real-world environment

Content

- Software testing life cycle
- 7 principles of software testing
- **Levels of software testing**
- Types of software testing

Types of software testing

- Functional testing
- Non-functional testing
- Structural testing
- Change-related testing

Functional testing

- Functional testing = Black-box testing
- Based on functional requirements
- Detect functional defects
- Don't care how to implement

Functional testing

- Black-box design techniques
  - Equivalence Partitioning
  - Boundary Value Analysis
  - State Transition Diagrams
  - Decision Tables
  - Cause-Effect Graph
  - Use Case Testing

Non-functional testing

- Performance testing
- Usability testing
- Security testing
- Configuration/Installation testing
- Back-up/Recovery testing

Non-functional testing

- Performance testing
  - Speed – Whether the application responds quickly
  - Scalability – The maximum user load the software application can handle
  - Stability – If the application is stable under varying loads

Non-functional testing

- Usability testing
  - Effectiveness
    - Is the system is easy to learn?
  - Efficiency
    - Little navigation should be required to reach the desired screen
    - Uniformity in the format of screen
  - Accuracy
    - No outdated or incorrect data should be present
    - No broken links should be present
  - User Friendliness
    - Controls used should be self-explanatory
    - Help should be provided for the users

Non-functional testing

- Security testing
  - Confidentiality – limiting access to sensitive access managed by a system
  - Integrity – ensuring that data is consistent, accurate, and trustworthy throughout its lifecycle and cannot be modified by unauthorized entities
  - Authentication – ensuring sensitive systems or data are protected by a mechanism that verifies the identity of the individual accessing them
  - Authorization – ensuring sensitive systems or data properly control access for authenticated users according to their roles or permissions
  - Availability – ensuring that critical systems or data are available for their users when they are needed
  - Non-repudiation – ensures that data sent or received cannot be denied, by exchanging authentication information with a provable time stamp

Non-functional testing

- Configuration/Installation testing
  - Configuration testing
    - Different hardware, environments
    - Software configuration
    - Version upgrade conflict
  - Installation testing
    - Install packages
    - Uninstall

Non-functional testing

- Back-up/Recovery testing
  - Verifies software's ability to recover from failures like software/hardware crashes, network failures

Structural testing

- Structural testing = White-box testing
- Design test cases based on source code
- Code coverage
  - Statement coverage
  - Decision coverage
  - Condition coverage
  - Path coverage
  - Loop coverage

Change-related testing

- Test after bugs are fixed
- Re-testing/Confirmation testing
  - Execute the exact test cases that found the bugs
  - Confirm the bugs have been fixed
  - No guarantee that new bugs have not occurred
- Regression testing
  - Execute all previously passed test cases
  - Find new bugs that occur


Q
A