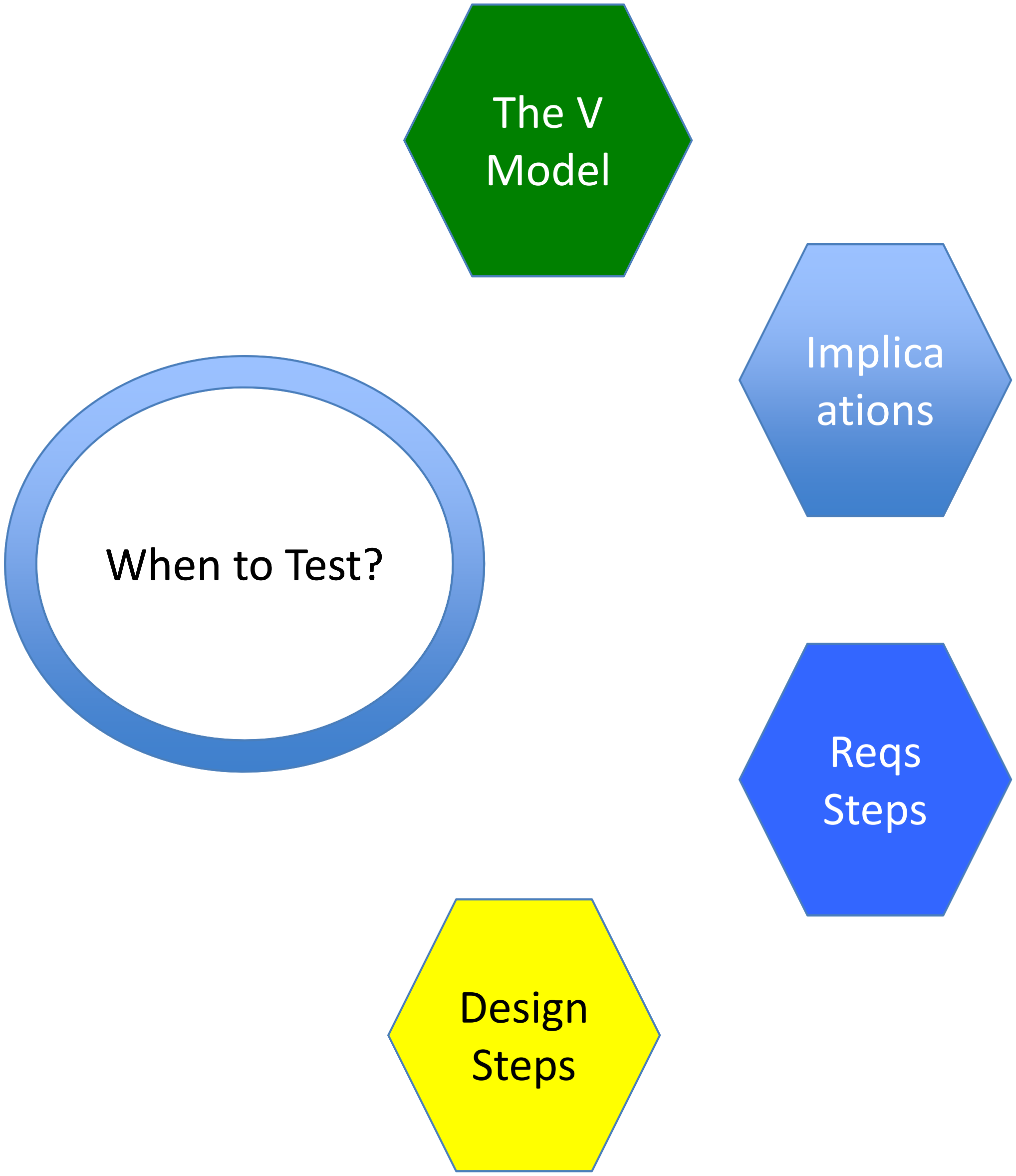
Integration Testing

Objectives

After this lesson, you should know about

a) The V-Model of development and its implications



When to Test?

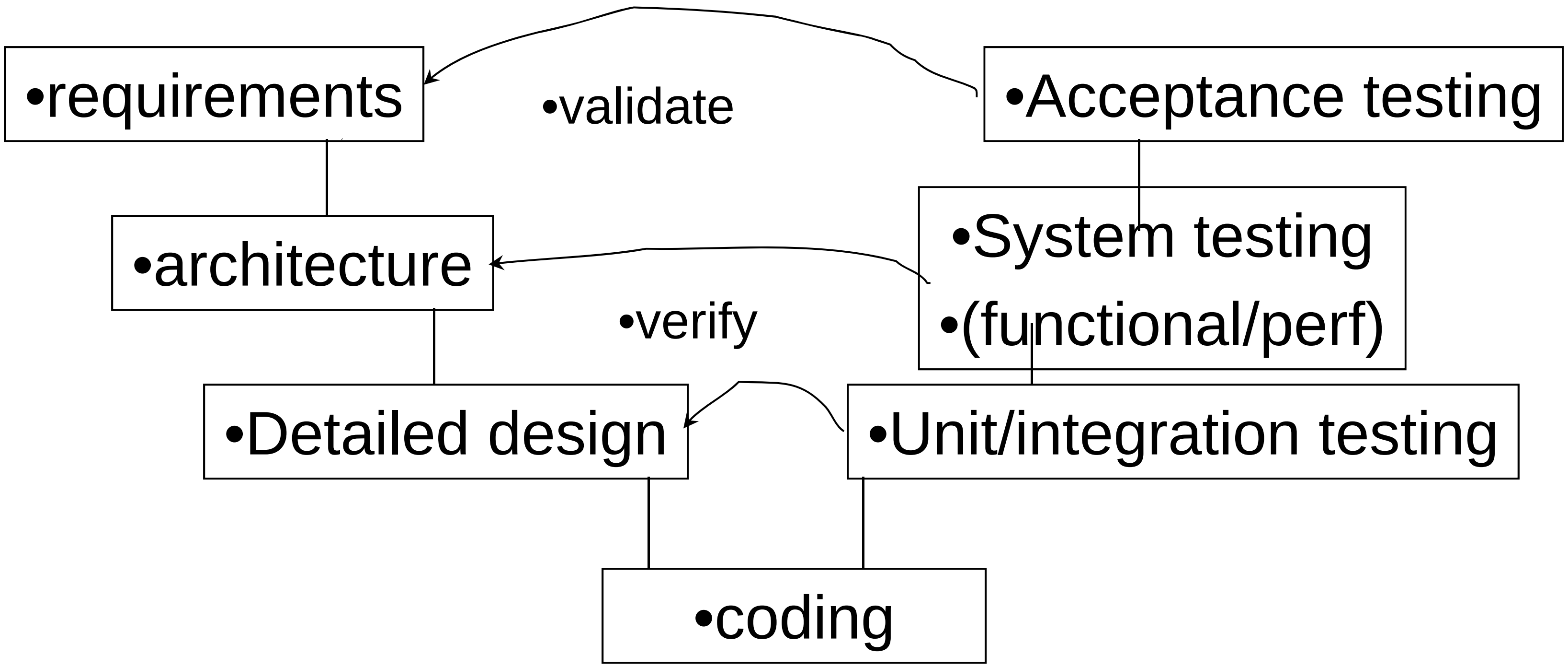
The V Model

Implicaations

Reqs Steps

Design Steps

The V model



•requirements

•validate

•Acceptance testing

•architecture

•System testing

•(functional/perf)

•verify

•Detailed design

•Unit/integration testing

•coding

Implications of V-Model

1. Errors in upstream processes are more expensive to debug and fix.

2. Not only that - Industry data also shows frequency of errors occuring in upstream processes is **higher**!

- Testers should be also involved in requirements and design phases of development.

- Inspections/reviews to TRAP errors from flowing "downstream" is essential.

What can you do during reqs?

- Validate
  - – Show prototypes/screensketches
  - – Design Fit-criterion and corresponding acceptance tests

- Verify (Inspect Requirements document)
  - Evaluate each requirement for correctness, ambiguity, testability, etc

What can you do during arch/coding?

- Design to be testable: Controllable/Observable
- Plan out top-down and other integration testing mechanisms
- Logging (levels of verbosity) - for debugging
- Checkpointing - for debugging
- pre-conditions, post-conditions, assertions

INTEGRATION TESTING

Big-Bang Integration

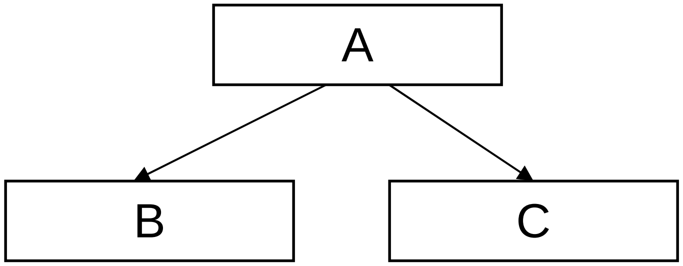
- After all components are unit tested we may test the entire system with all its components in action.
- **(-) may be impossible to figure out where faults occur unless faults are accompanied by component-specific error messages**

Bottom-Up Integration Testing

- **Bottom-Up Integration**: each component at lower hierarchy is tested individually; then the components that rely upon these are tested.

- **Driver:** a routine that simulates a call from parent component to child component

Bottom-Up Testing Example



1) Test B, C individually (using drivers)

2) Test A such that it calls B
If an error occurs we know that the problem is in A or in the interface between A and B

3) Test A such that it calls C
If an error occurs we know that the problem is in A or in the interface between A and C

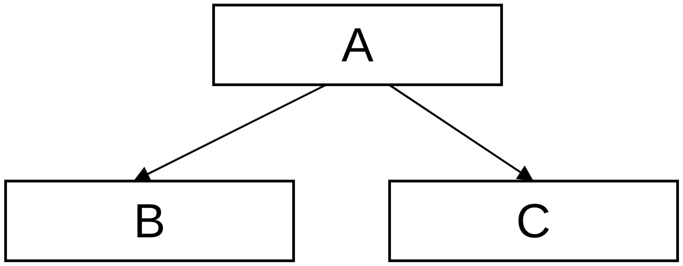
(-) Top level components are the most important yet tested last.

Top-Down Integration Testing

- **Top-Down Integration**: each component at higher position in hierarchy is tested individually; then the components that they rely upon are tested.

- **Stub:** a routine that fakes behavior of a child component.

Top-Down Testing Example



1) Test A individually (use stubs for B and C)

2) Test A such that it calls B (stub for C)
If an error occurs we know that the problem is in B or in the interface between A and B

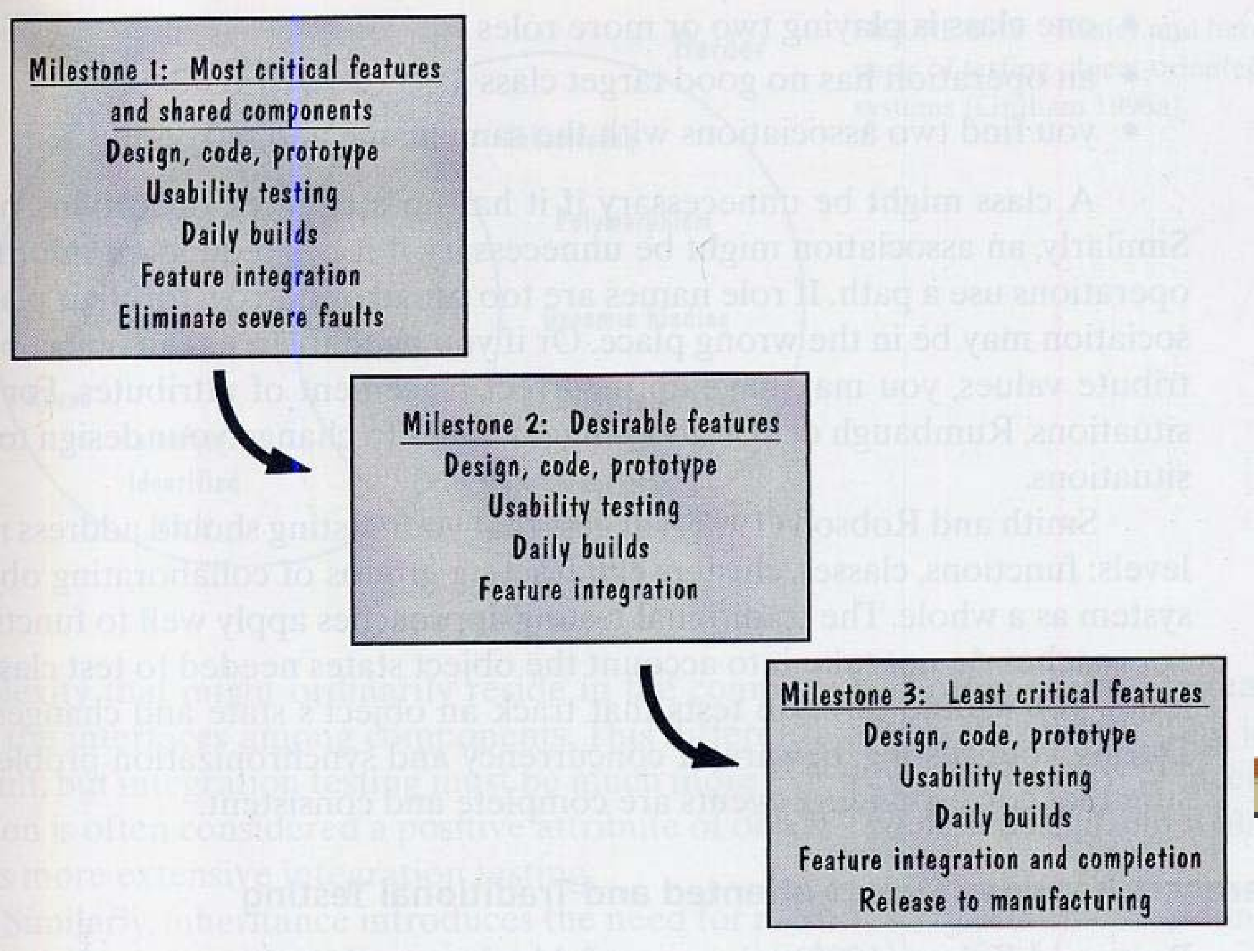
3) Test A such that it calls C (stub for B)
If an error occurs we know that the problem is in C or in the interface between A and C

* Stubs are used to simulate the activity of components that are not currently tested; (-) may require many stubs

Sandwich Integration

- Multi-level component hierarchy is divided into three levels with the test target being in the middle:
- Top-down approach is used in the top layer;
- Bottom-down approach used in the lower layer.

Sync & Stabilize Approach



Milestone 1: Most critical features and shared components

Design, code, prototype
Usability testing
Daily builds
Feature integration
Eliminate severe faults

Milestone 2: Desirable features

Design, code, prototype
Usability testing
Daily builds
Feature integration

Milestone 3: Least critical features

Design, code, prototype
Usability testing
Daily builds
Feature integration and completion
Release to manufacturing

MOCKITO

Mockito

- A testing tool that helps to create stubs and to verify that calls are made.

- when some method is called - do something.
- verify that some methods were called (to test interactions between methods).

- Examples on Eclipse..