



# **SENG3320/6320:** **Software Verification and Validation**

**School of Electrical Engineering and Computing**

**Semester I, 2020**

# Non-Functional Testing

# More Types of Testing

- Performance testing
  - Performance testing is generally executed to determine how a system performs in terms of responsiveness and stability under a particular workload.
- Stress testing
  - Stress testing is a way to test reliability under unexpected or rare workloads.
- Security testing
  - Security testing is concerned mainly with the security-related aspects of the software.
- Usability testing
  - Usability testing is concerned mainly with the use of the software.

# Performance Testing

- **Performance testing** evaluates system performance under normal and heavy usage.
- Website performance is crucial to the success of any web application.
- Performance testing considers scalability and load of the system
  - Test the website's ability to handle the real-world volumes.
  - Typically by generating many user access simulations.

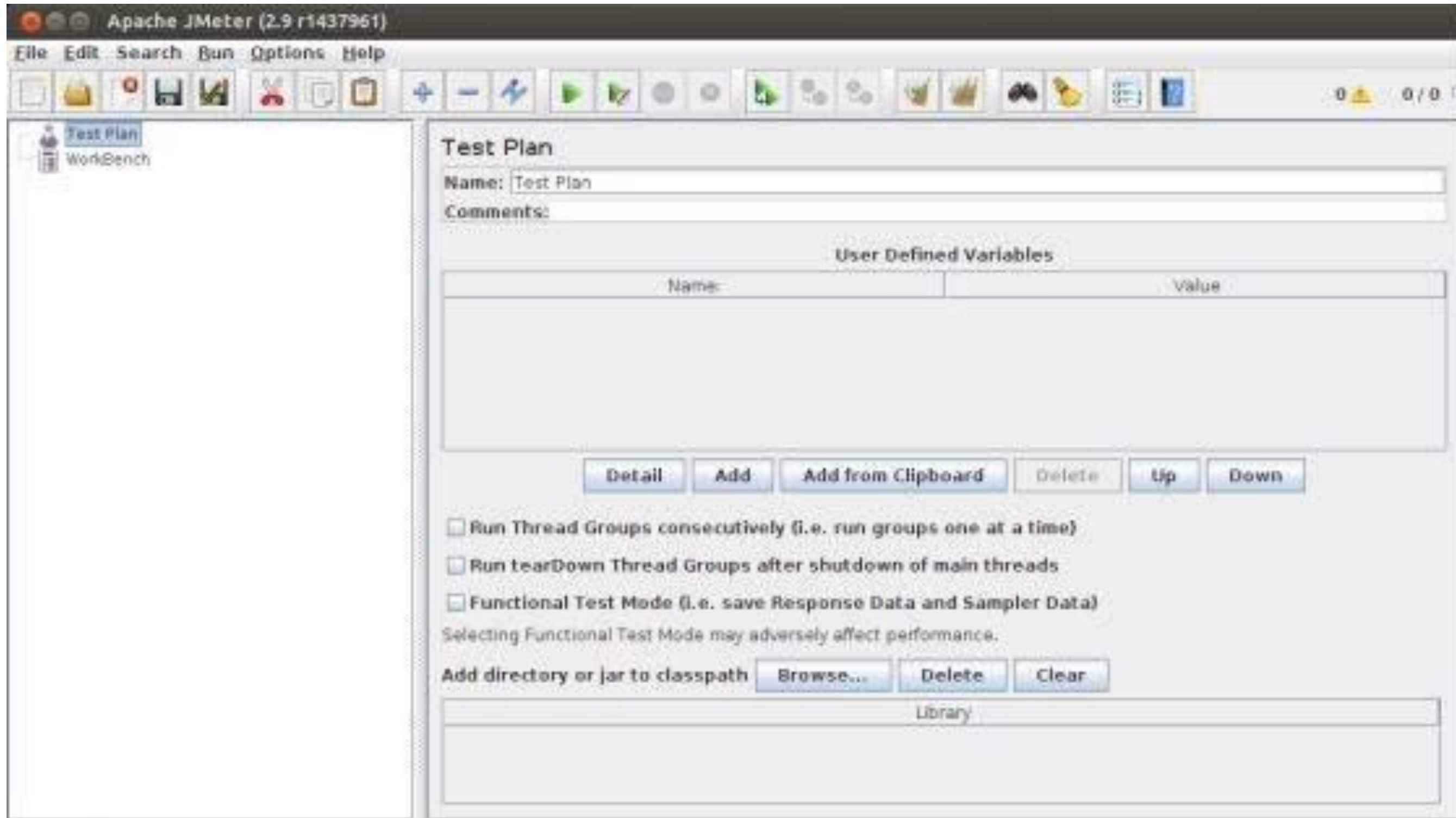
# Performance Testing

- JMeter (<https://jmeter.apache.org/>)
- A Java application designed to load test functional behavior and measure performance.
- JMeter simulates a group of users sending requests to a target server, and returns statistics that show the performance/functionality of the target server/application via tables, graphs, etc.



# Performance Testing

- JMeter (<https://jmeter.apache.org/>)



# Stress Testing

- **Stress testing** consists of subjecting the system to varying and maximum loads to evaluate the resulting performance.
- Stress testing can be automated. Tools can report the following type of information:
  - Number of requests, transactions and kilobyte/second
  - Round trip time (time from the user makes a request to the time that the users receives the result)
  - Number of concurrent connections
  - Degradation of performance
  - Types of visitors to the site and their number
  - CPU and memory usage of the application server

# Security Testing

- Security is a primary concern when communicating and conducting business especially sensitive and business critical transactions over the internet.
- Regardless whether the application requires the user to enter a password to access the website, the tester must check for internet threats.



# Top 2 Web Application Security Risks

- **Injection** – Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query.
- **Cross Site Scripting (XSS)** – XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping.

# Injection - Vulnerability

System is vulnerable when user input is passed without tests

```
String query = "select * from user where username=";  
query += req.getParameter("userID");  
query += "' and password = '"+req.getParameter("pwd")+"''";
```

```
SELECT * FROM user WHERE username=' OR 'a'='a' --AND password=''';
```

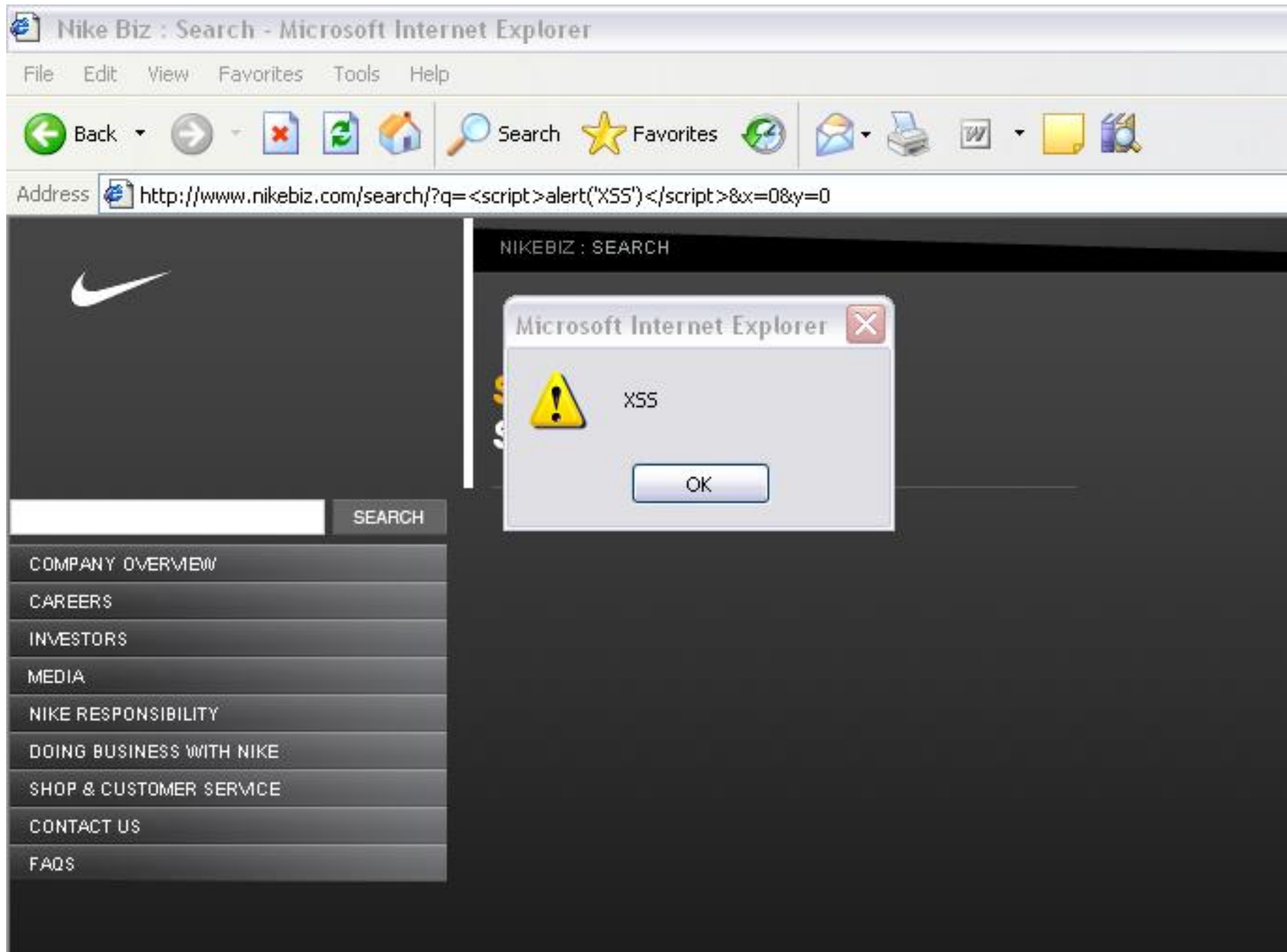
# Injection - Protection

- Validate Input
  - Careful with special characters such as < > " ' and =
  - Whitelist (e.g. /[a-zA-Z0-9]{0,20}/)
  - Validate length, type, and syntax
- Avoid the use of interpreter if possible
  - Use stored procedures
- Otherwise: Use safe APIs
  - Strongly typed parameterized queries (such as PreparedStatement)
- Use an ORM (Object Relational Manager)
  - such as Hibernate or Spring

# XSS - Vulnerability

- Exploit URL:
  - `http://www.nikebiz.com/search/?q=<script>alert('XSS')</script>&x=0&y=0`
- HTML returned to victim:
  - `<div id="pageTitleTxt"> <h2><span class="highlight">Search Results</span><br />Search: "<script>alert('XSS')</script>"</h2>`

# XSS - Vulnerability



# XSS – Protection

- Appropriate encoding of all output data.
  - HTML or XML depending on output mechanism
  - means `<script>` is encoded `&lt;script&gt;`;
  - encode all characters other than a very limited subset
  - specify the character encoding (e.g. ISO 8859-1 or UTF 8)
- How To: Prevent Cross-Site Scripting in ASP.NET

<http://msdn.microsoft.com/en-us/library/ms998274.aspx>

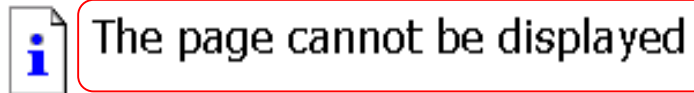
- XSS Prevention Cheat Sheet:

[http://www.owasp.org/index.php/XSS\\_%28Cross\\_Site\\_Scripting%29\\_Prevention\\_Cheat\\_Sheet](http://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet)

# Usability Testing



- **Usability testing** assesses the website's user friendliness and suitability by gathering information about how users interact with the site.
- The key to usability testing is to study what a user actually does.
- Usability testing steps:
  - Identify the website's purpose
  - Identify the intended users
  - Define tests and conduct the usability testing
  - Analyze the acquired information

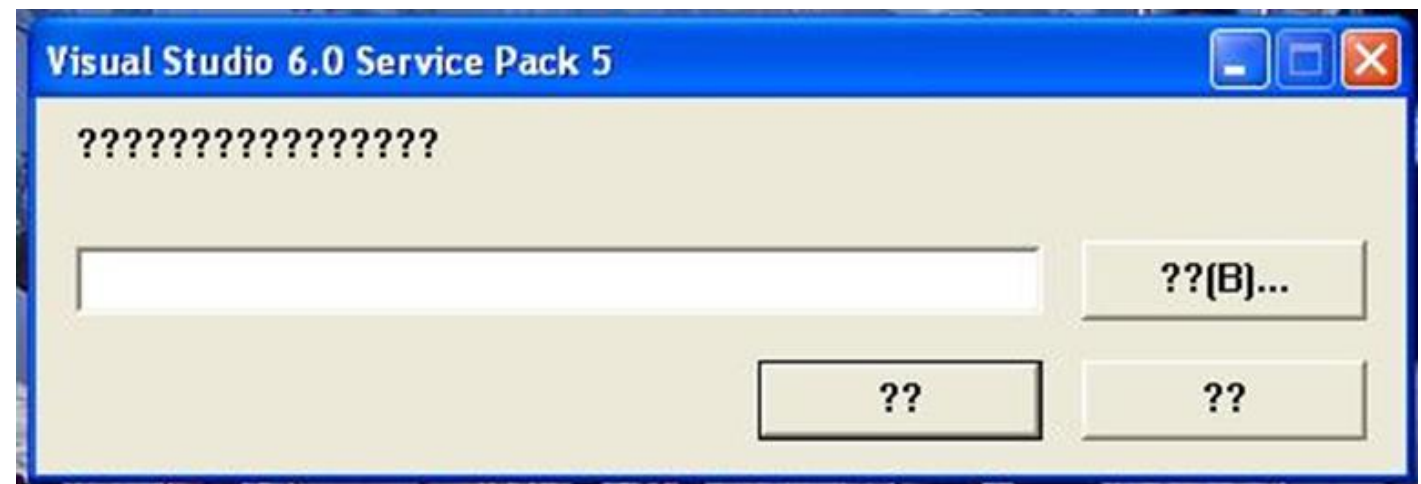
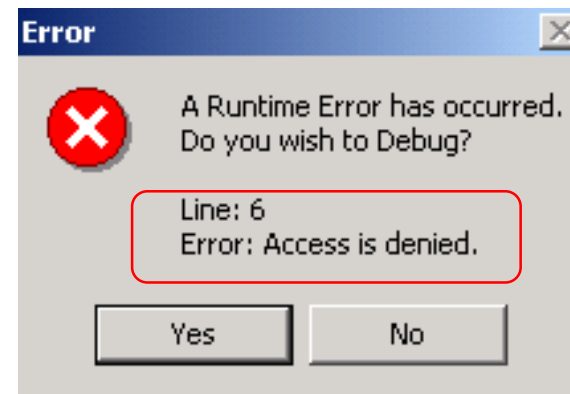
# Usability Examples



The page you are looking for is currently unavailable. The Web site might be experiencing technical difficulties, or you may need to adjust your browser settings.

Please try the following:

- Click the  [Refresh](#) button, or try again later.
- If you typed the page address in the Address bar, make sure that it is spelled correctly.
- To check your connection settings, click the **Tools** menu, and then click **Internet Options**. On the **Connections** tab, click **Settings**. The settings should match those provided by your local area network (LAN) administrator or Internet service provider (ISP).
- If your Network Administrator has enabled it, Microsoft Windows can examine your network and automatically discover network connection settings. If you would like Windows to try and discover them, click  [Detect Network Settings](#)





# Compatibility Testing

- A key challenge in web applications is ensuring that the user sees a web page as the designer intended:
  - The user can select different browser software and browsers options.
  - Use different network software and online service
  - Run concurrent applications
- **Compatibility testing** ensures product functionality and reliability on the supported browsers and platforms that exist on the customer computer.

# Compatibility Testing

- Guideline for testing web applications (by listing the platform and browser environments to be tested).

Table 7.5 Browser compatibility table

Browser Platform	Netscape Communicator 4.5	Netscape Communicator 4.7	America Online 4.0	Internet Explorer 4.01	Internet Explorer 5.0	...
Windows 95						
Windows 98						
Windows NT						
Windows 2000						
Windows ME						
Mac OS X						
iMac						
Macintosh 8.1						
Linux				not applicable	not applicable	
...						

# Resources

- A Guide for Building Secure Web Applications and Web Services

[https://www.owasp.org/index.php/Category:How\\_To](https://www.owasp.org/index.php/Category:How_To)

- Advanced SQL Injection in SQL Server Applications

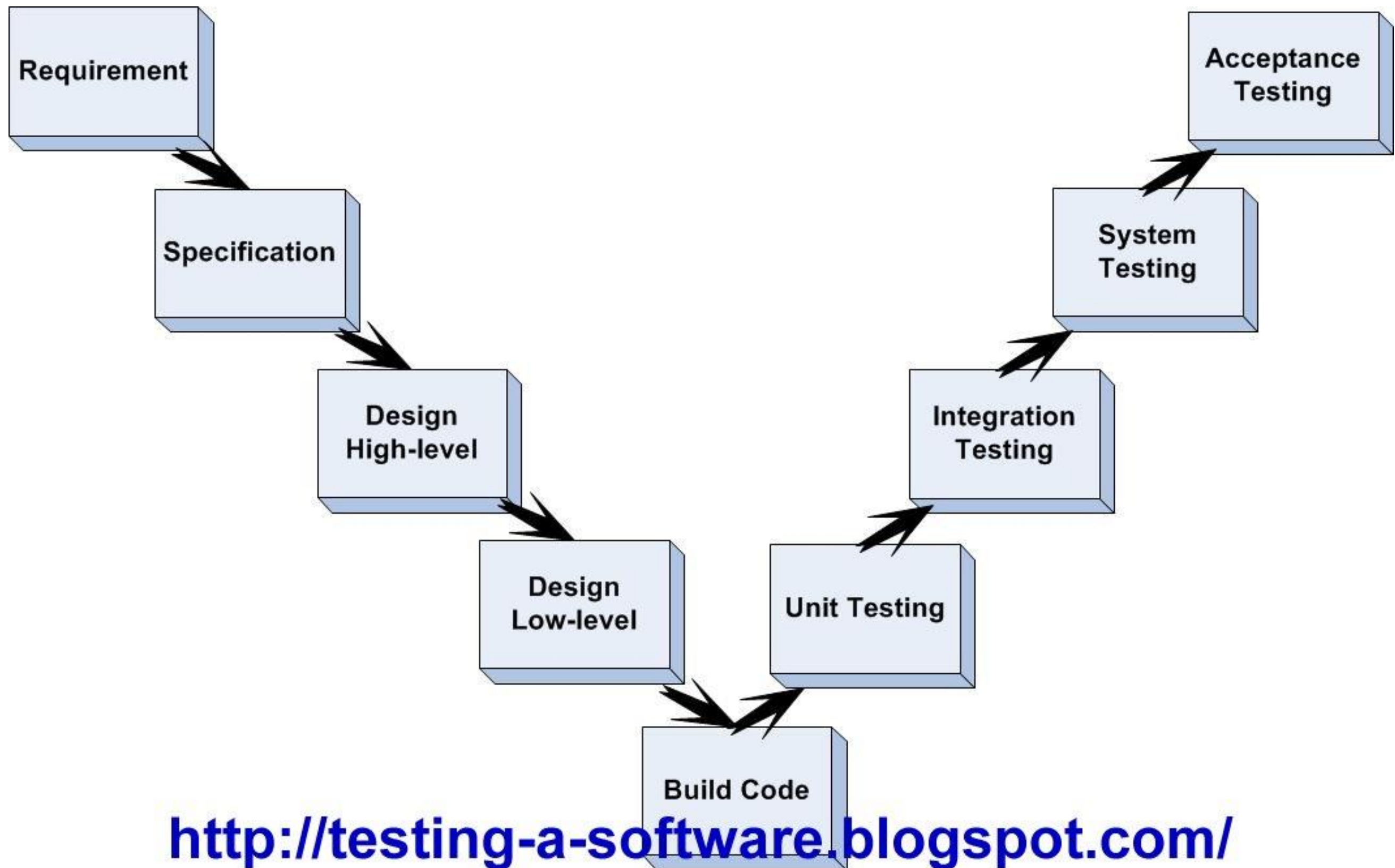
<https://sparrow.ece.cmu.edu/group/731-s11/readings/anley-sql-inj.pdf>

- Testing Guide:

- [https://www.owasp.org/images/5/56/OWASP\\_Testing\\_Guide\\_v3.pdf](https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf)
- [https://www.owasp.org/index.php/Web\\_Application\\_Security\\_Testing\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Web_Application_Security_Testing_Cheat_Sheet)

# Testing Lifecycle

# The V-model of development



# Types of Testing

- Unit (Module) testing
  - testing of a single module in an isolated environment
- Integration testing
  - testing parts of the system by combining the modules
- System testing
  - testing of the system as a whole after the integration phase
- Acceptance testing
  - testing the system as a whole to find out if it satisfies the requirements specifications

# Unit Testing

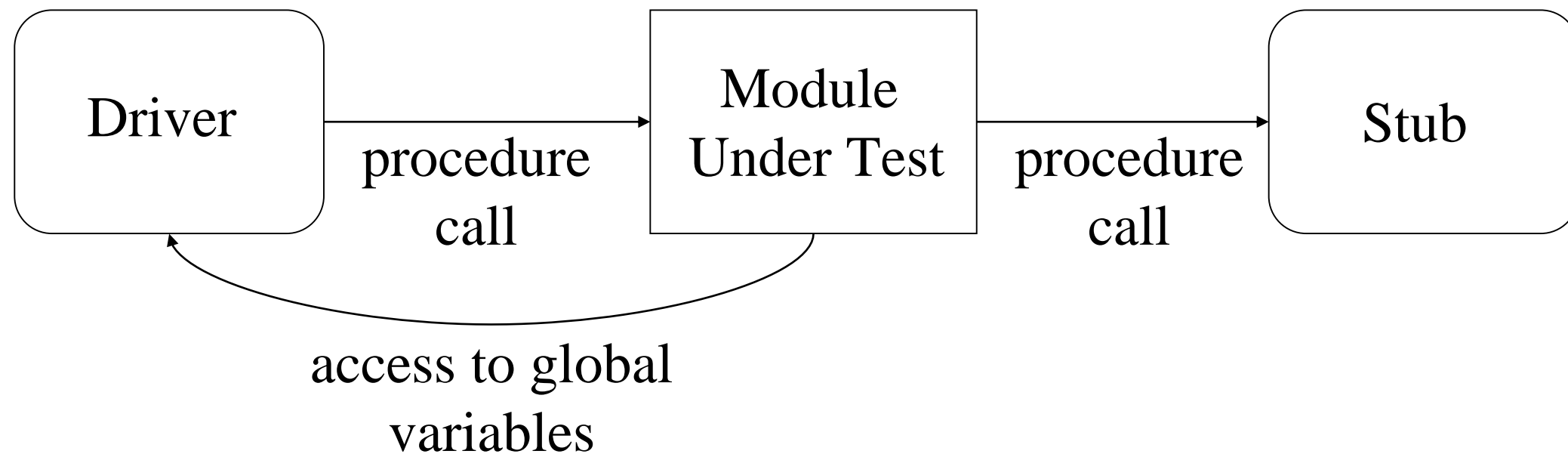
- Involves testing a single isolated module
- Note that unit testing allows us to isolate the errors to a single module
  - we know that if we find an error during unit testing it is in the module we are testing
- Modules in a program are not isolated, they interact with each other. Possible interactions:
  - calling procedures in other modules
  - receiving procedure calls from other modules
  - sharing variables
- For unit testing we need to isolate the module we want to test, we do this using two things
  - drivers and stubs

# Drivers and Stubs

- **Driver:** A program that calls the interface procedures of the module being tested and reports the results
  - A driver simulates a module that calls the module currently being tested
- **Stub:** A program that has the same interface as a module that is being used by the module being tested, but is simpler.
  - A stub simulates a module called by the module currently being tested
- **Mock objects:** Create an object that mimics only the behavior needed for testing



# Drivers and Stubs

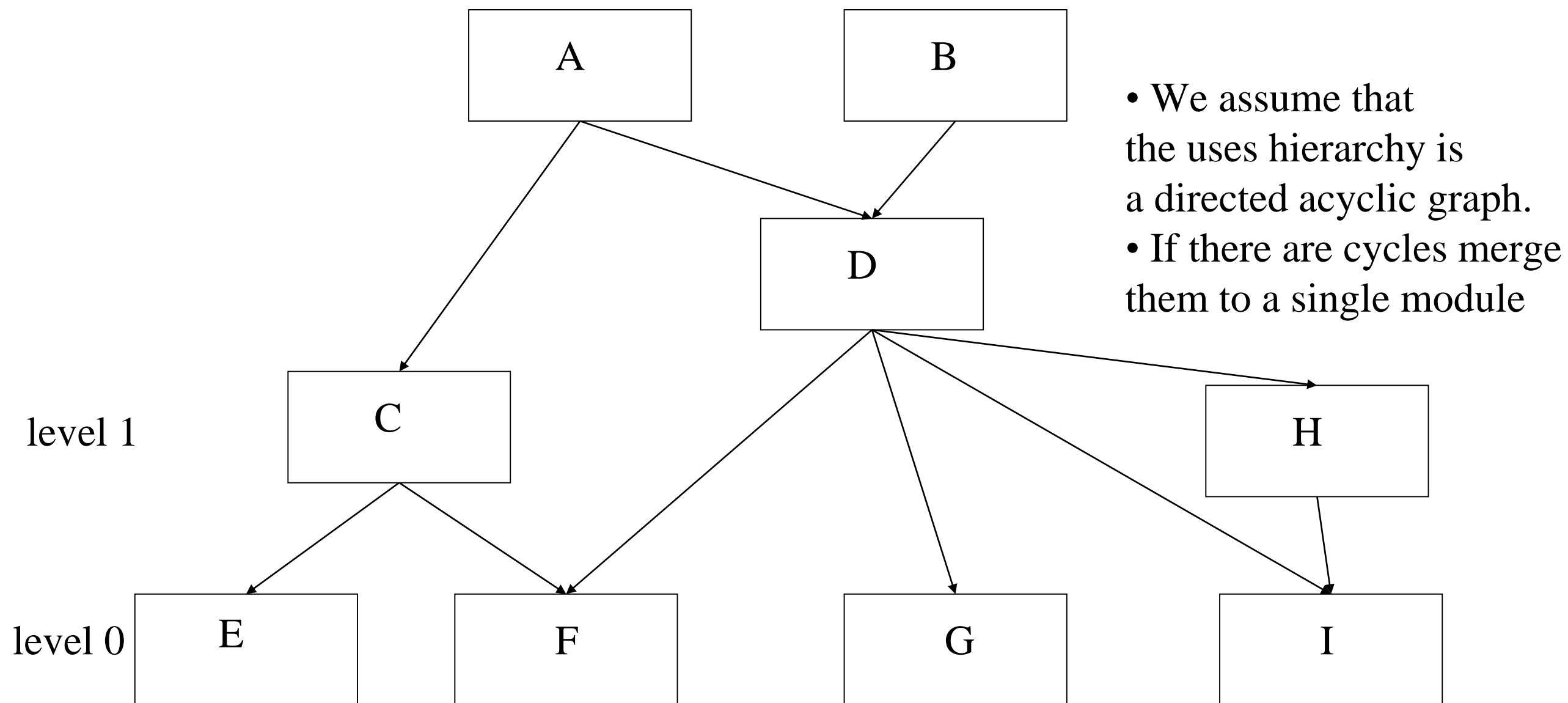


- Driver and Stub should have the same interface as the modules they replace
- Driver and Stub should be simpler than the modules they replace

# 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

# 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

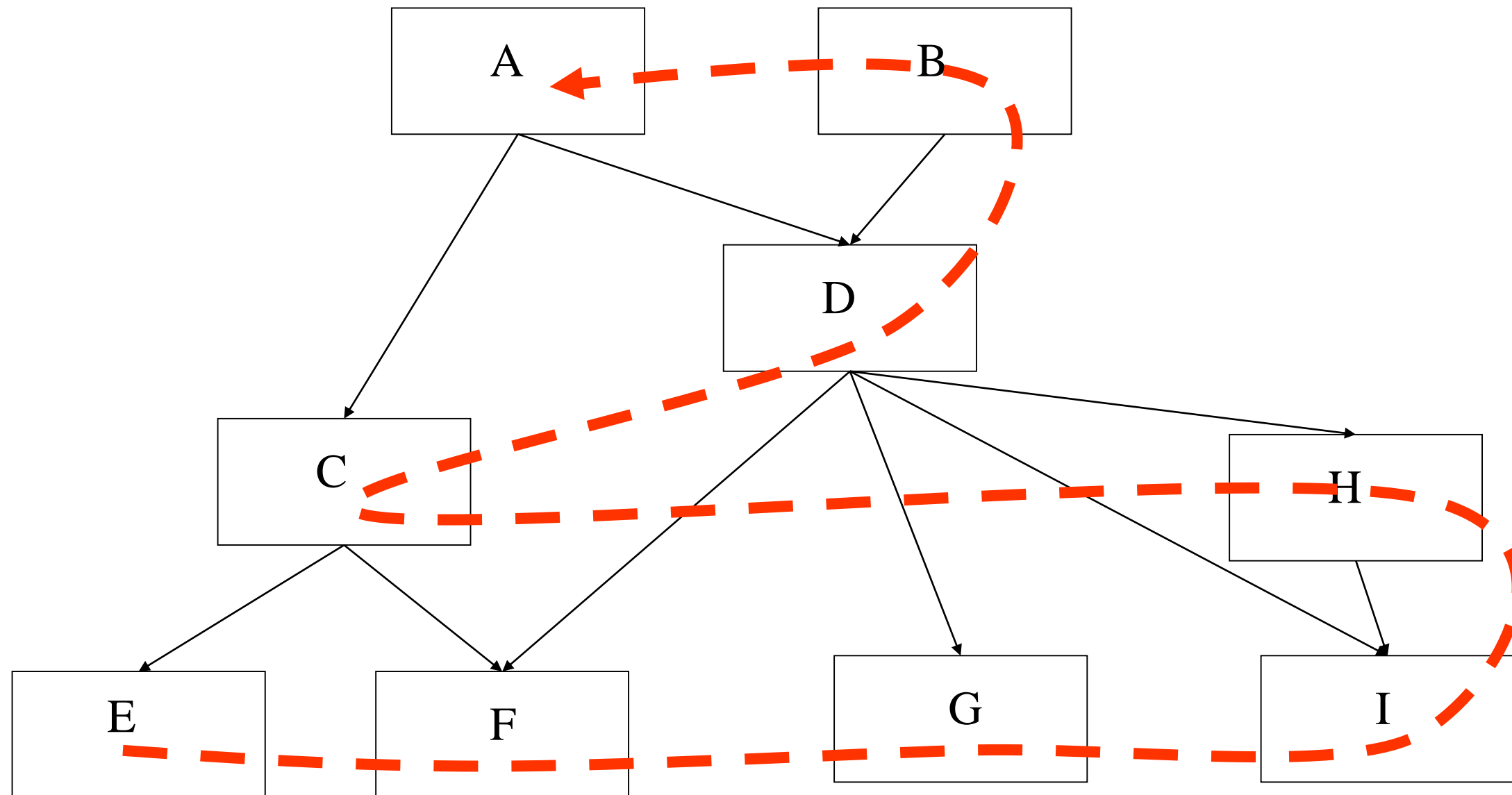
• level 0 components do not use any other components

• level  $i$  components use at least one component on level  $i-1$  and no component at a level higher than  $i-1$

# Bottom-Up Integration

- 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

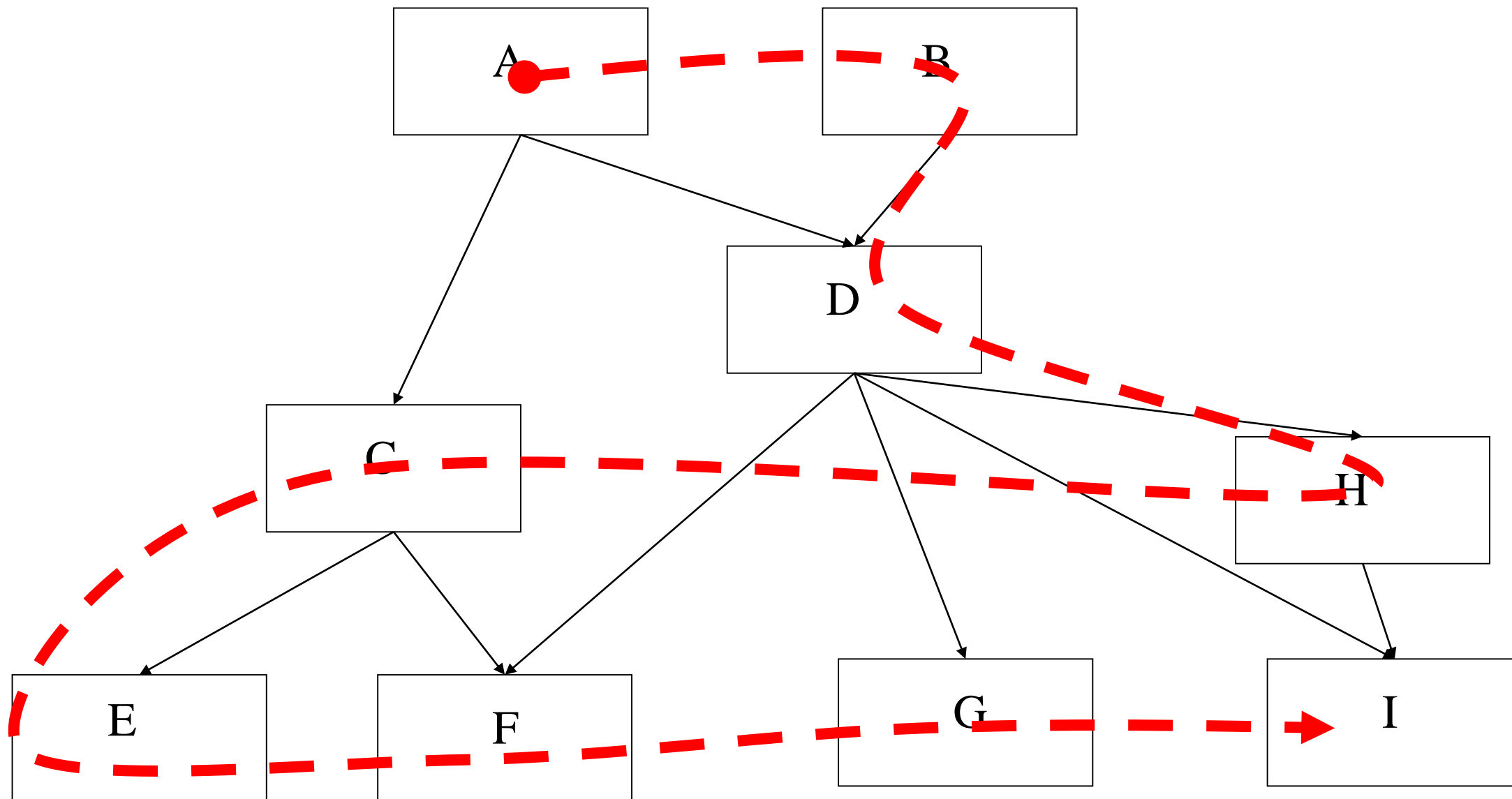
# Bottom-up Integration



# Top-down Integration

- 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



# Other Approaches to Integration

- 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/Acceptance Testing

- System testing/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
- Alpha and Beta testing
  - Alpha testing is performed within the development organization
  - Beta testing is performed by a select group of friendly customers

