Introduction to Software Engineering

(CS350)

Lecture 16

Jongmoon Baik

KOREA ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY
KAIST
한국과학기술원
SINCE 1971

Software Testing Strategy

What is Software Testing?

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.

"[**Software testing**] is the design and implementation of a special kind of software system: one that exercises another software system with the intent of **finding bugs**."

Robert V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools* (1999)

What is Software Testing? (Cont.)

- Software Testing typically involves:
  - Execution of the software with representative Inputs under an actual operational conditions
  - Comparison of predicted outputs with actual outputs
  - Comparison of expected states with resulting states
  - Measurement of the execution characteristics (execution time, memory usage, etc)

Goals of Software Testing

- Immediate goal
  - Identify discrepancies b/w actual results and expected behavior
- Ultimate goal
  - Demonstrate conformance to specification
    - Guaranteeing correctness is not realistic in general.
  - Provide an indication of correctness, reliability, safety, security, performance, usability, etc.
- Practical goal
  - Improve software reliability at an acceptable cost
    - Increase its usefulness

Software Testing Issues

Is an exhaustive test possible?

How much testing is required?

How to find a few input that represent the entire input domain?

When we can stop testing?

How to generate test data?

Which procedure is proper?

What is S/W Testing Strategy?

"An elaborate and systematic plan of series of actions to be taken for software testing to locate defects and remove them before the software system is released"

"No Universal Testing Strategy"

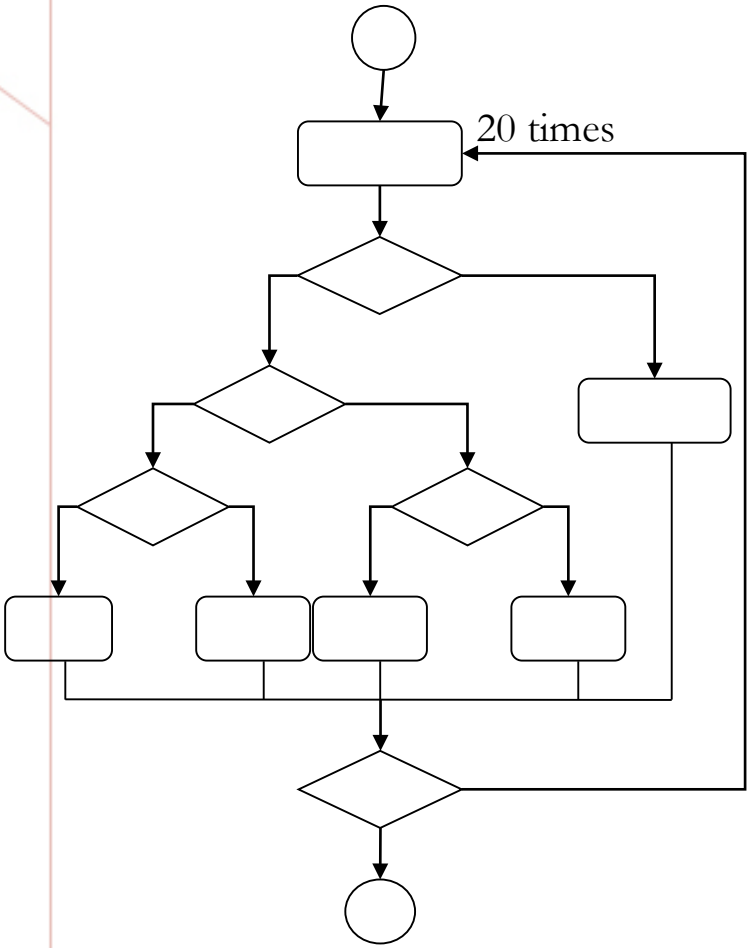
Strategic Approach

- To perform effective testing, you should conduct effective technical reviews. By doing this, many errors will be eliminated before testing commences.
- Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
- Different testing techniques are appropriate for different software engineering approaches and at different points in time.
- Testing is conducted by the developer of the software and (for large projects) an independent test group.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

Test Case, Test Oracle, & Test Bed

- Test Case:
  - A test related items which contains the following information
    - A set of test inputs
    - Execution conditions
    - Expected outputs
- Test Oracle:
  - A document, or piece of software that allows tester to determine whether a test has been passed or failed
    - e.g.: a specification (pre- and post conditions), a design document, and a set of requirements
- Test Bed
  - An environment that contains all the hardware and software needed to test a software component or a software system

Exhaustive Testing



- How many possible paths?
  - About 10^{14} paths
- Assume that one test case/millisecond
- How long it will take to execute all the test cases?
  - 3170 years (24 hrs, 7days working)

Go with
SELECTIVE TESTING

Questions to be Answered

- How do you conduct the tests?
- Should you develop a formal plan for your tests? If then When?
- Should you test the entire program as whole or run tests only on a small part of it?
- Should you rerun tests you've already conducted as you add new components to a large system?
- When should you involve the customer?

A Way of Thinking about Testing

- Constructive tasks – Software Analysis, Design, Coding, Documentation
- Destructive tasks – Testing
  - Considered to break the software
- Developers treats lightly, designing and executing tests that will demonstrate that the program works, rather than finding errors

V & V

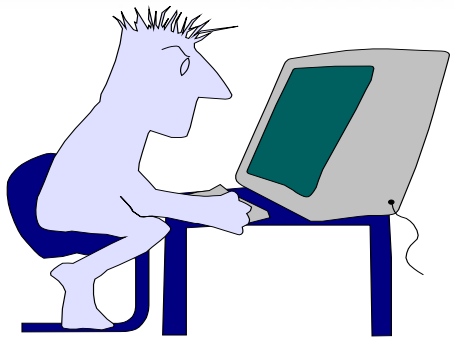
- *Verification* refers to the set of tasks that ensure that software correctly implements a specific function.
- *Validation* refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. Boehm [Boe81] states this another way:
  - *Verification:* "Are we building the product right?"
  - *Validation:* "Are we building the right product?"

Who Tests the Software?

developer

Understands the system
but, will test "gently"
and, is driven by "delivery"

- Responsible for testing the individual unit of the program
- Also conduct the integration testing
- Correct uncovered errors



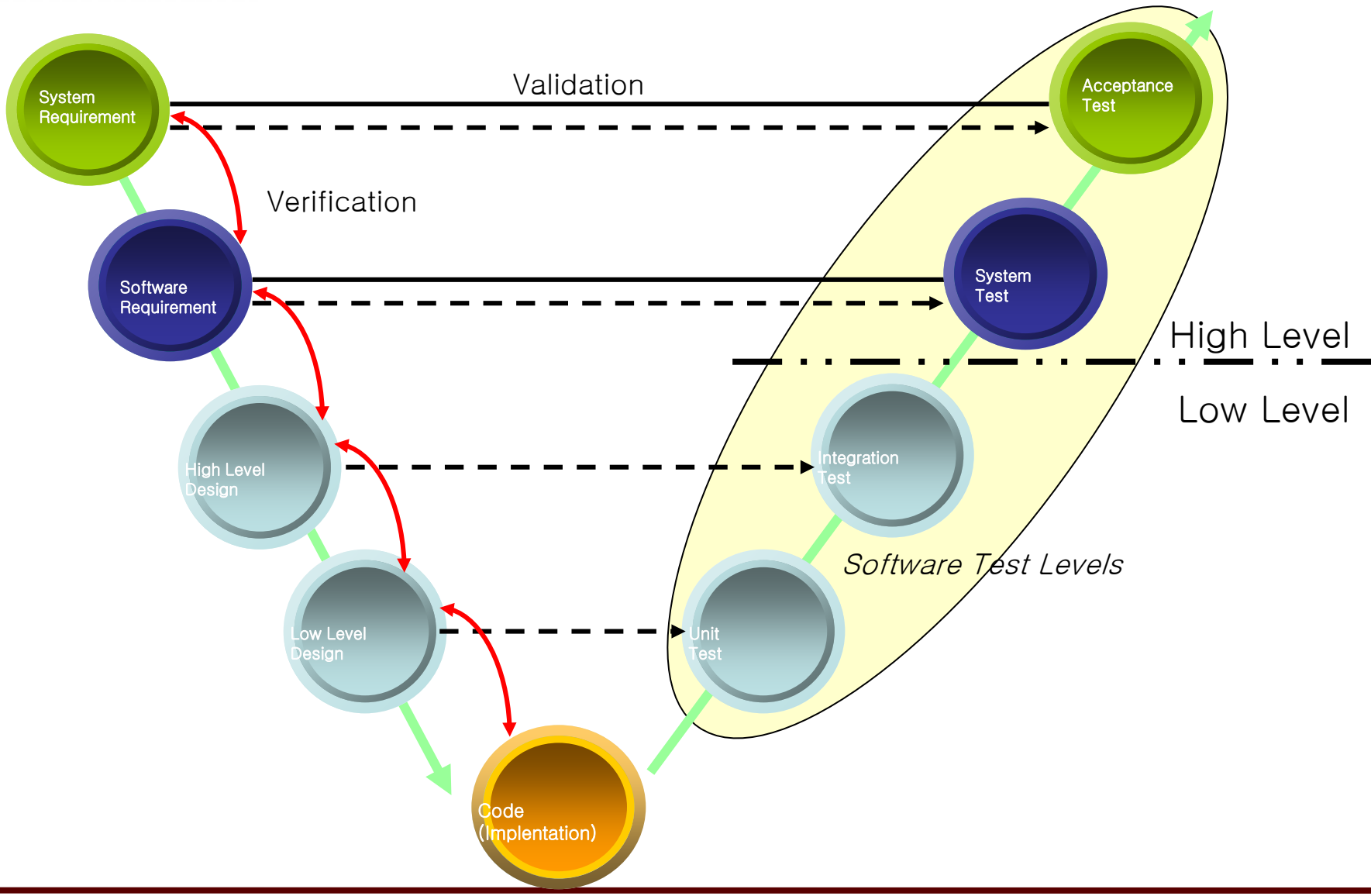
independent tester

Must learn about the system,
but, will attempt to break it
and, is driven by quality

- Remove the inherent problems of developers' testing their programs
- Report to SQA organization : Independence

Close Collaboration b/w Development Group and ITG are required for successful tests throughout the life Cycle

Testing Strategy



Testing Strategy – cont.

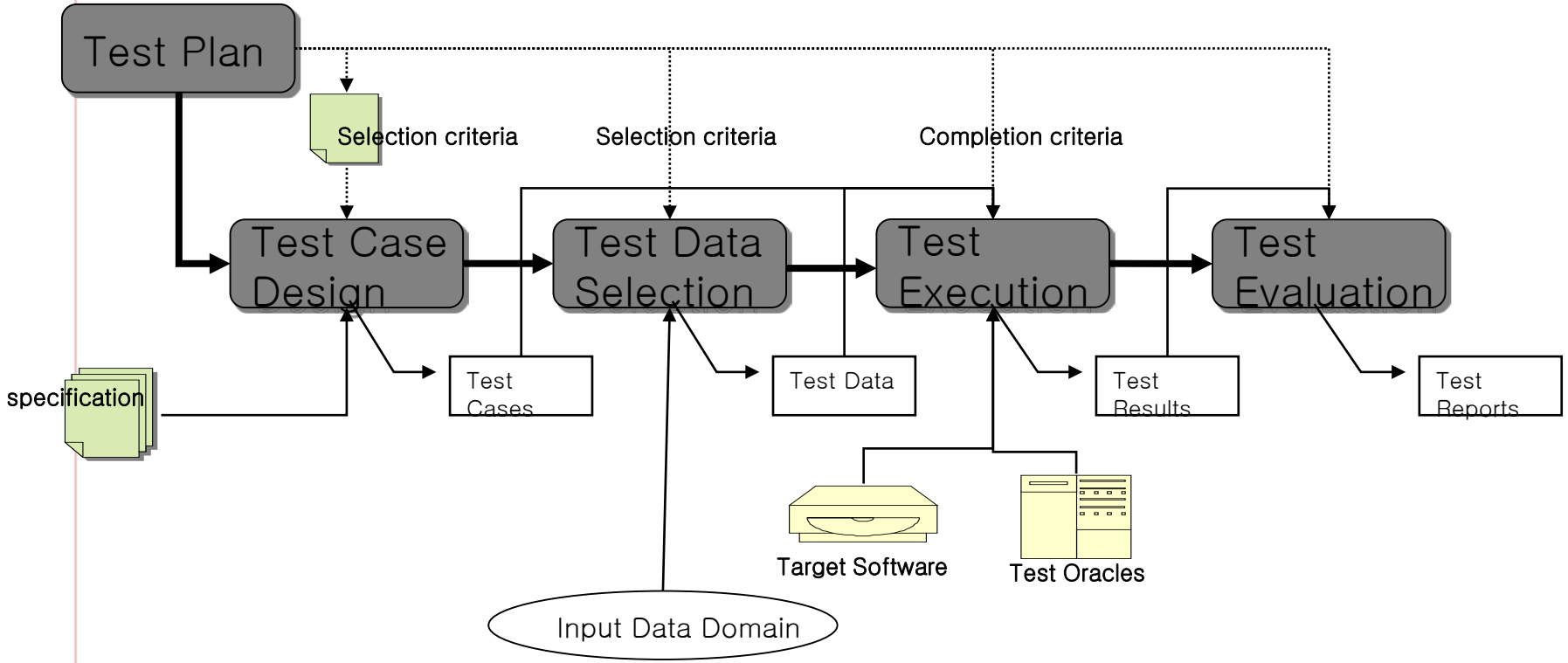
- We begin by ‘testing-in-the-small’ and move toward ‘testing-in-the-large’
- For conventional software
  - The module (component) is our initial focus
  - Integration of modules follows
- For OO software
  - our focus when “testing in the small” changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

Strategic Issues

- Specify product requirements in a quantifiable manner long before testing commences.
- State testing objectives explicitly.
- Understand the users of the software and develop a profile for each user category.
- Develop a testing plan that emphasizes “rapid cycle testing.”
- Build “robust” software that is designed to test itself
- Use effective technical reviews as a filter prior to testing
- Conduct technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

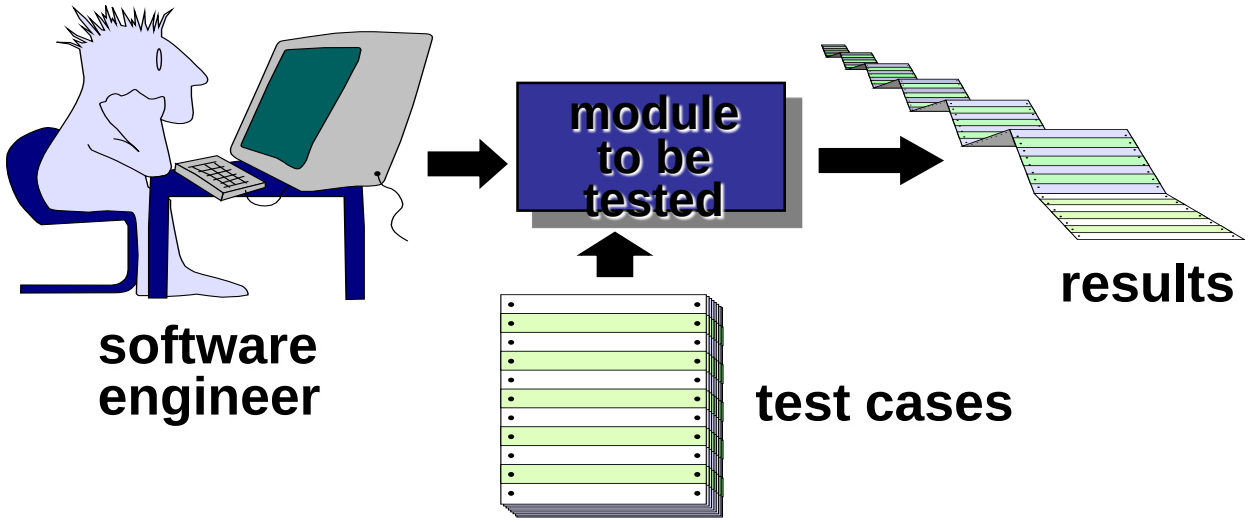
Software Test Process

- Included in a entire software development process
- Test plan start with requirement analysis
- Test plan & Test procedure: systematic and performed in parallel with s/w development



Unit (Component) Testing

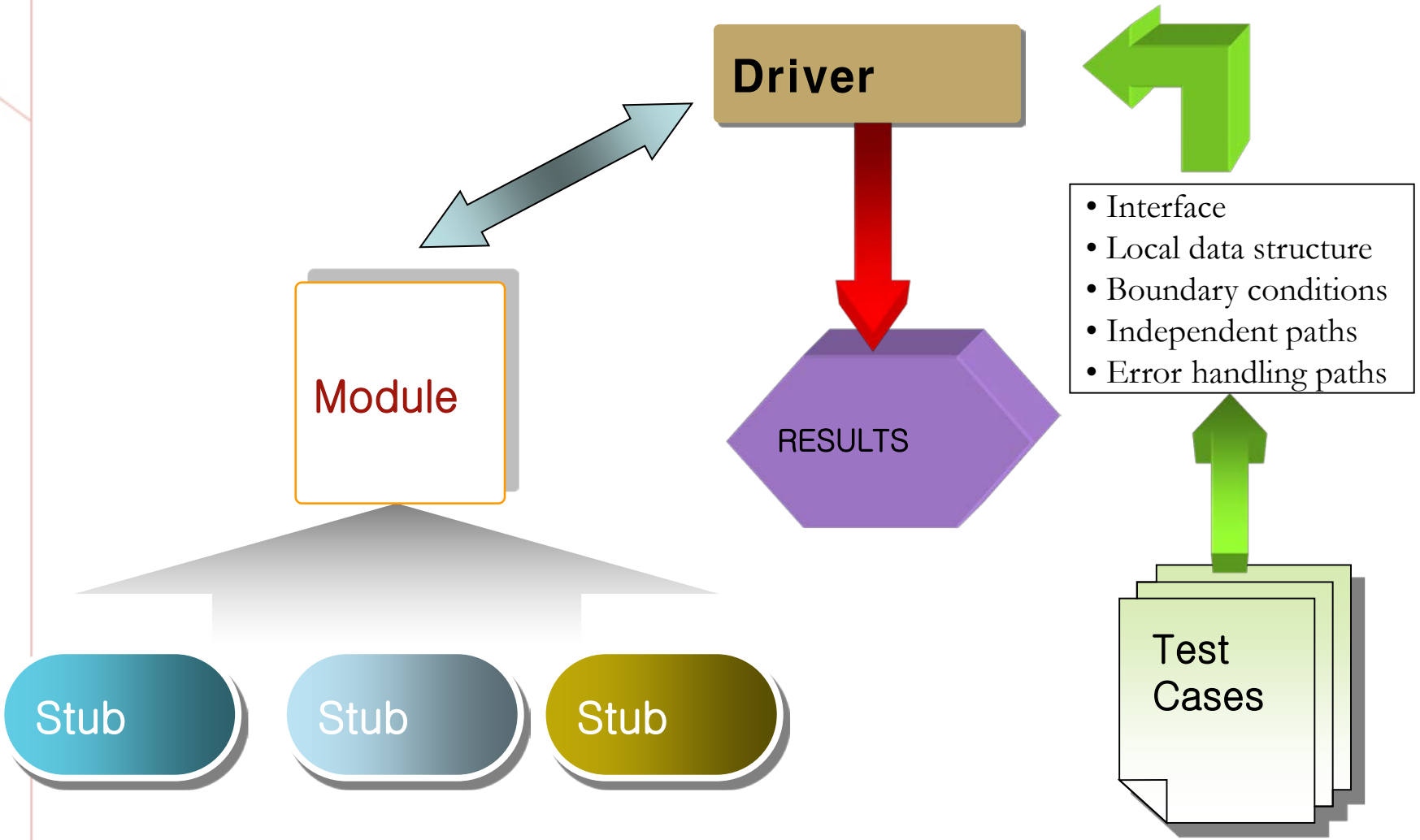
- Focus on the smallest software design (module or component)
- Often corresponds to the notion of “compilation unit” from the prog. Language
- Responsibility: Developer
- Test internal processing logic and data structure within the boundary of a component
- Can be conducted in parallel for multiple components
- May be necessary to create stubs: “fake” code that replaces called modules
  - If not yet implemented, or not yet tested



What are tested in Unit Testing?

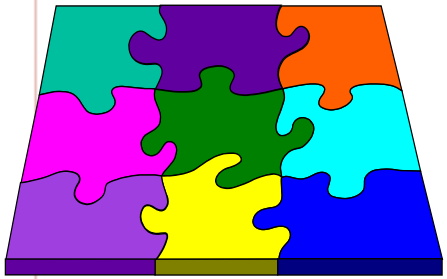
- Information flows for module interfaces
- Local data structures
- All independent paths (basis path) through control structure
- Boundary condition
- Error handling paths

Unit Test Environment



Integration Testing

- Exercise two or more combined units (or components)
- Main objectives:
  - Detect interface errors
  - Assure the functionalities when combined
- Responsibility: Developers or Testing Group



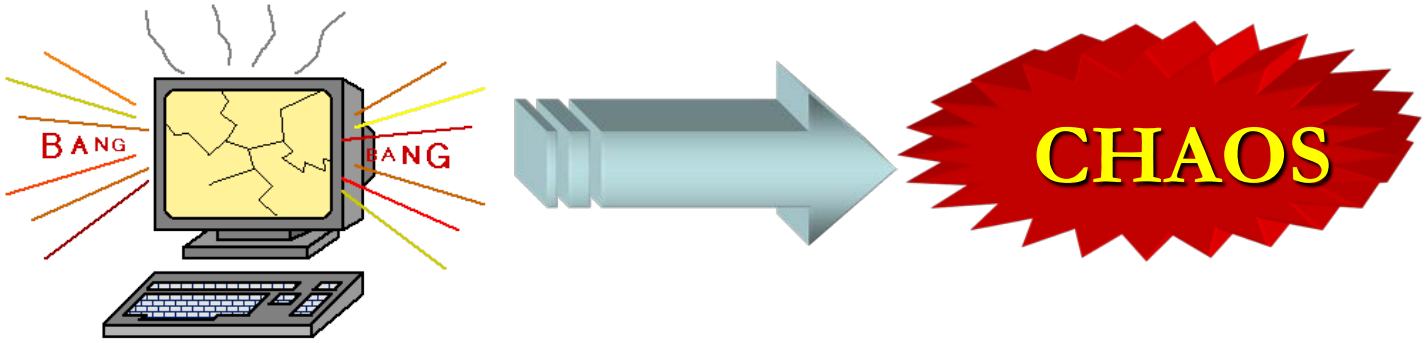
- Issues
  - Integration Strategy (How to Combine?)
  - Integration with thirty-party components
    - Compatibility, Correctness, etc

Integration Testing Strategies

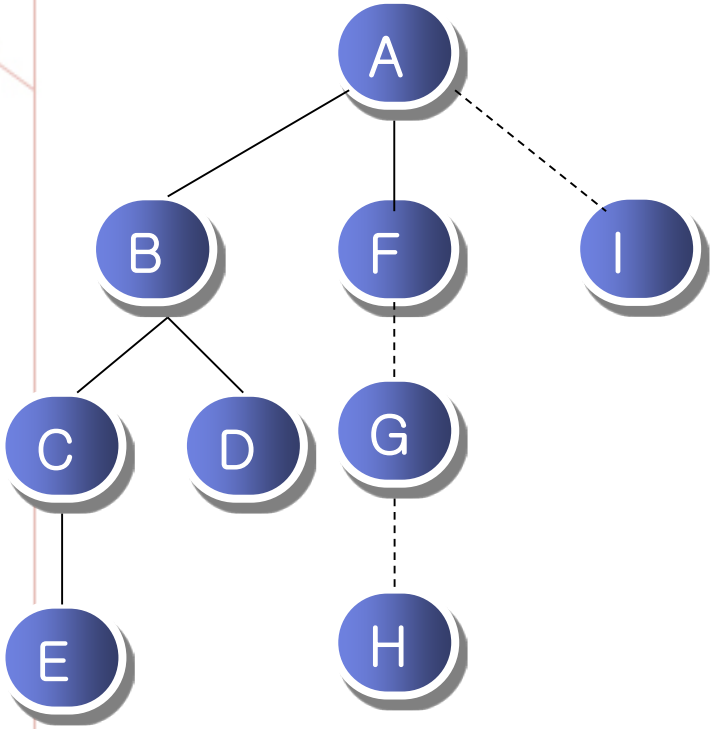
- Non- Incremental Integration
  - "Big Bang" approach
- Incremental Approaches
  - Top-down Integration
  - Bottom-up Integration
  - Sandwich Testing

"Big Bang" Approach

- All unit tested components are combined at once and tested as whole
- Disadvantages
  - Difficult to correct defects
    - Critical and peripheral modules not distinguished
  - When errors are corrected, new ones appear : endless loop
    - User does not see the product until very late in the development life cycle



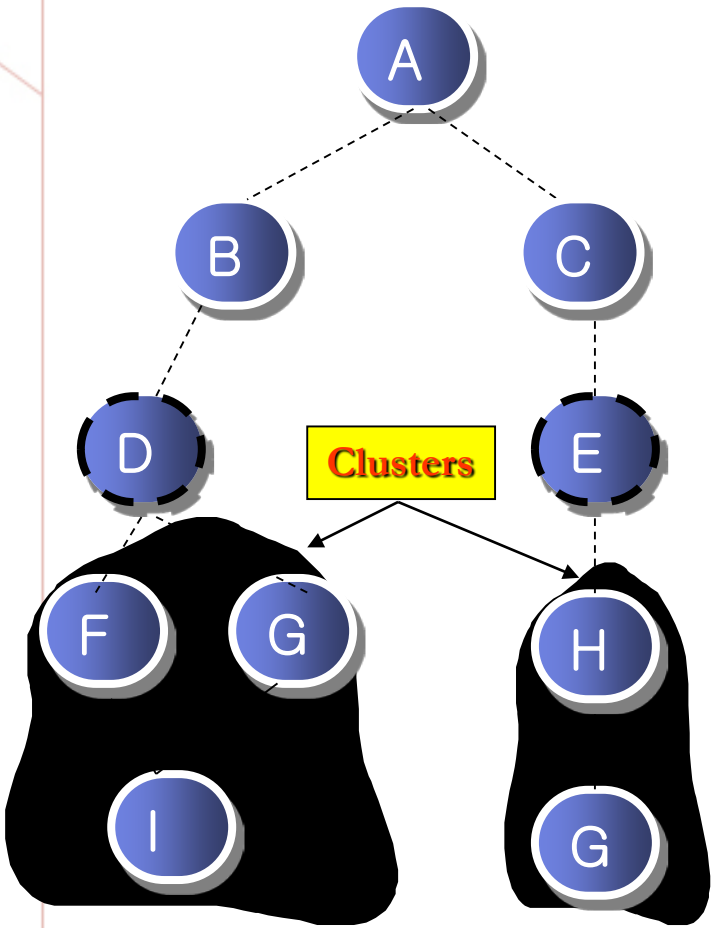
Top-down Integration



Depth-First Approach

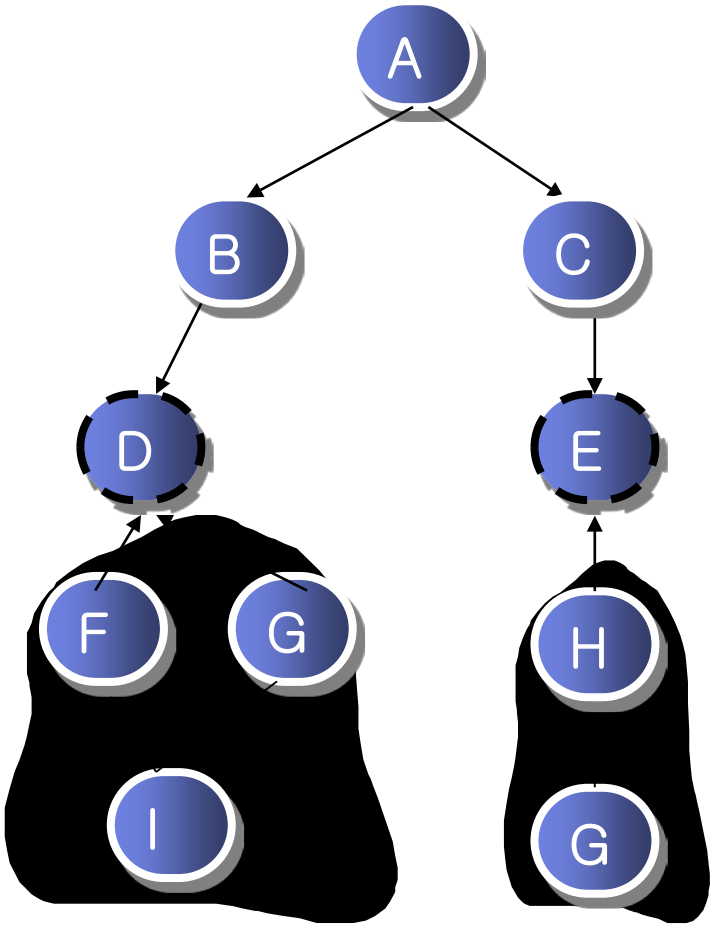
- An Incremental Approach to construction of the software architecture
- Integrated by moving downward through the control hierarchy
  - Depth-First or Breadth-First
- Begins with main control module (main program)

Bottom-up Integration



- Begins construction and testing with atomic modules
  - From components at the lowest levels in the program structure
- No need for stubs
- Drivers are replaced one at a time

Sandwich Integration



- Top level modules are tested with stubs
- Worker modules are integrated into clusters
- Advantages:
  - Significantly Reduced number of drivers
  - Simplified integration of clusters

Regression Testing

- *Regression testing* is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects

- Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed.

- Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors.

- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.

Smoke Testing

- A common approach for creating “daily builds” for product software
- Smoke testing steps:
  - Software components that have been translated into code are integrated into a “build.”
    - A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
  - A series of tests is designed to expose errors that will keep the build from properly performing its function.
    - The intent should be to uncover “show stopper” errors that have the highest likelihood of throwing the software project behind schedule.
  - The build is integrated with other builds and the entire product (in its current form) is smoke tested daily.
    - The integration approach may be top down or bottom up.

How to select a strategic option?

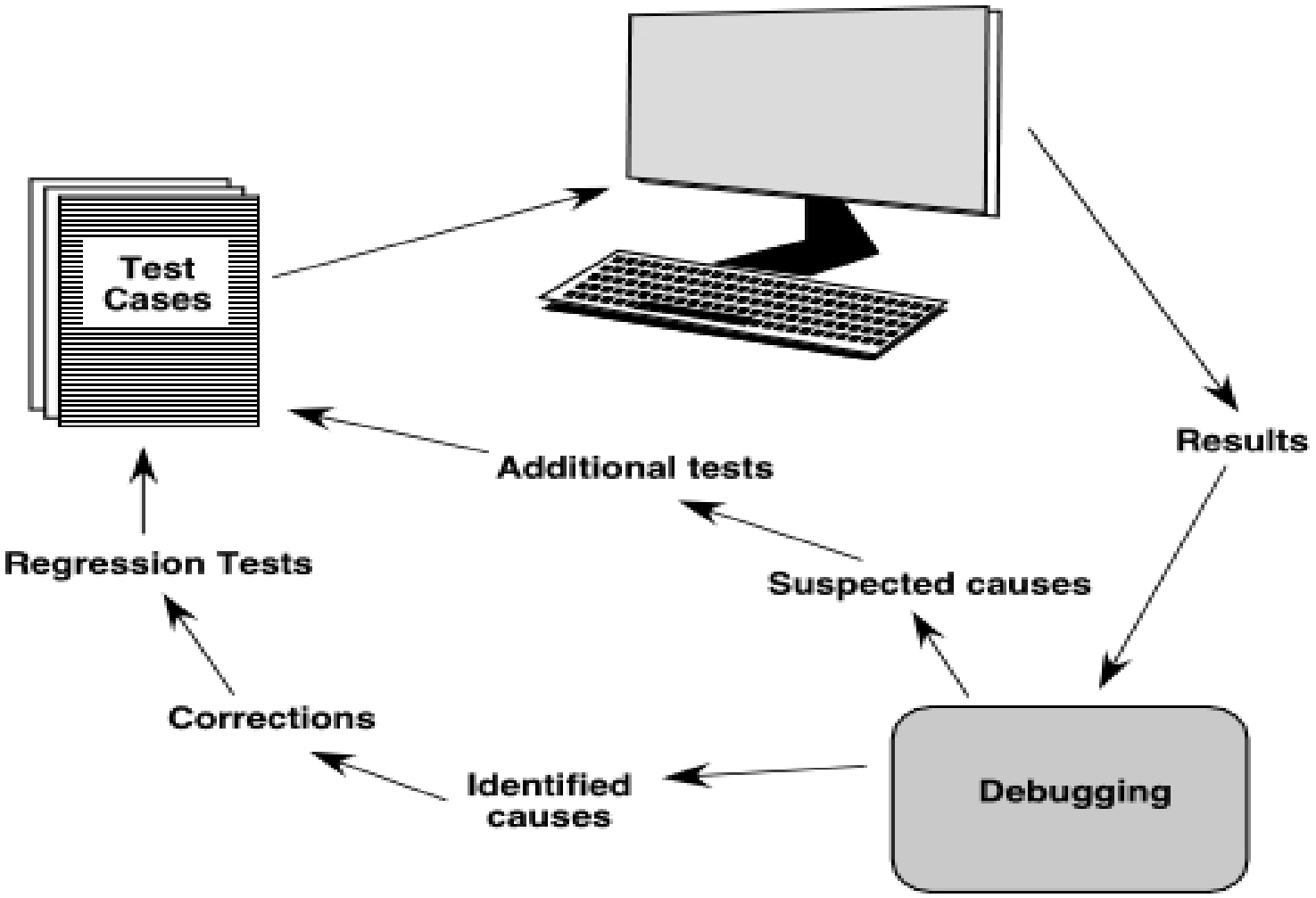
- Depends upon software characteristics and project schedule.
- Identify critical modules and test them as early as possible
  - Critical Module's characteristics:
    - Address several software requirements
    - Has a high level of control
    - Complex and error-prone
    - Has definite performance requirements
- Focus on critical module functions in regression tests

High Order Testing

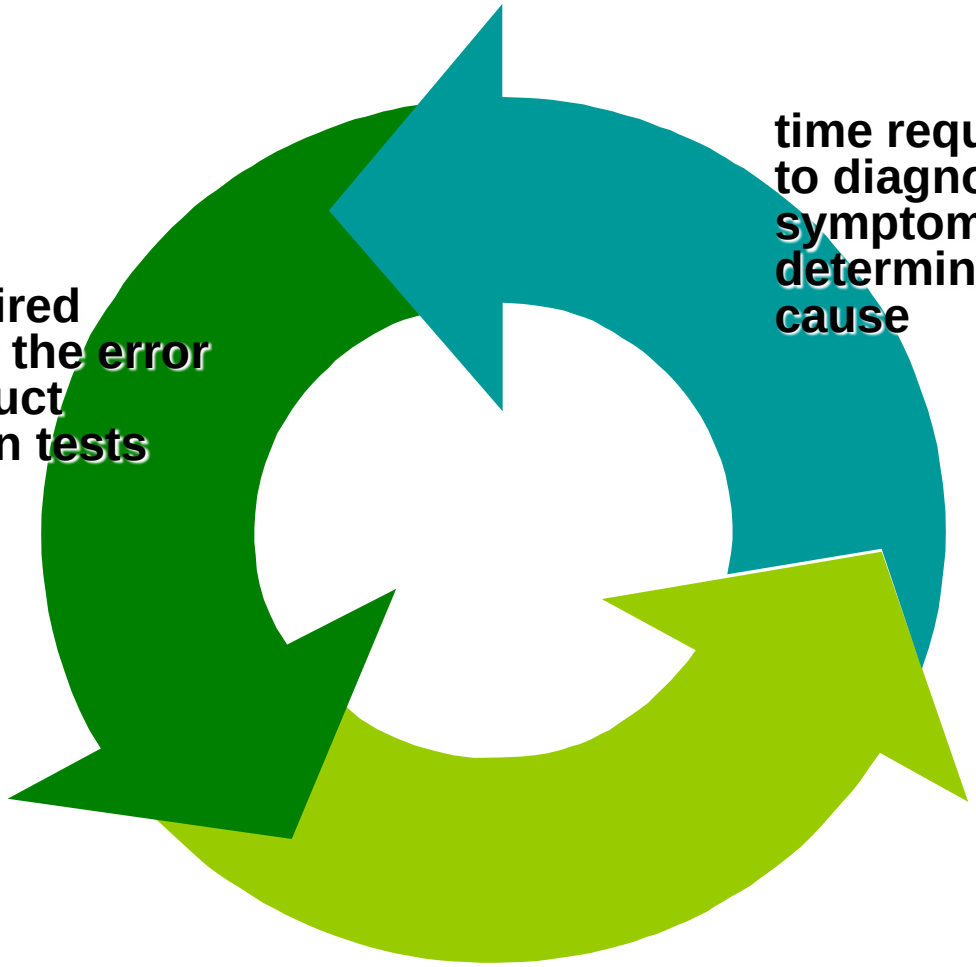
- Validation testing
  - Focus is on software requirements
- System testing
  - Focus is on system integration
- Alpha/Beta testing
  - Focus is on customer usage
- Recovery testing
  - forces the software to fail in a variety of ways and verifies that recovery is properly performed
- Security testing
  - verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration
- Stress testing
  - executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
- Performance Testing
  - test the run-time performance of software within the context of an integrated system

Debugging: A Diagnostic Process

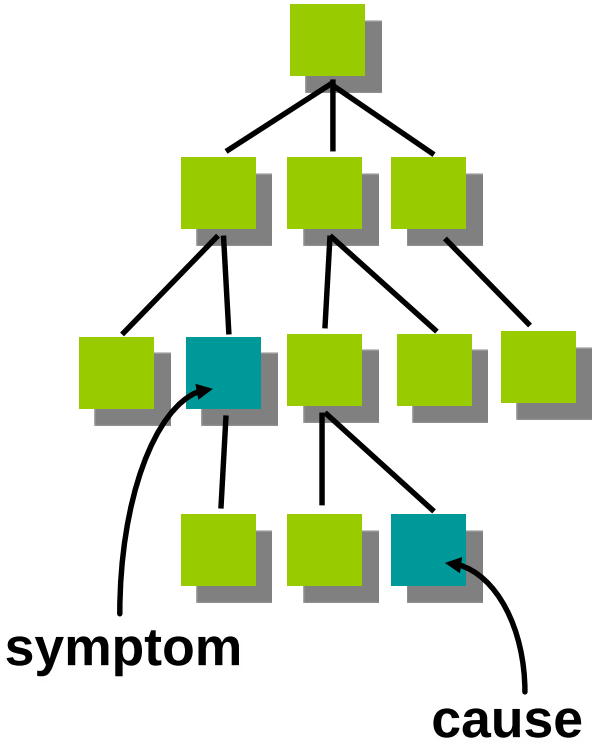
The Debugging Process



Debugging Effort

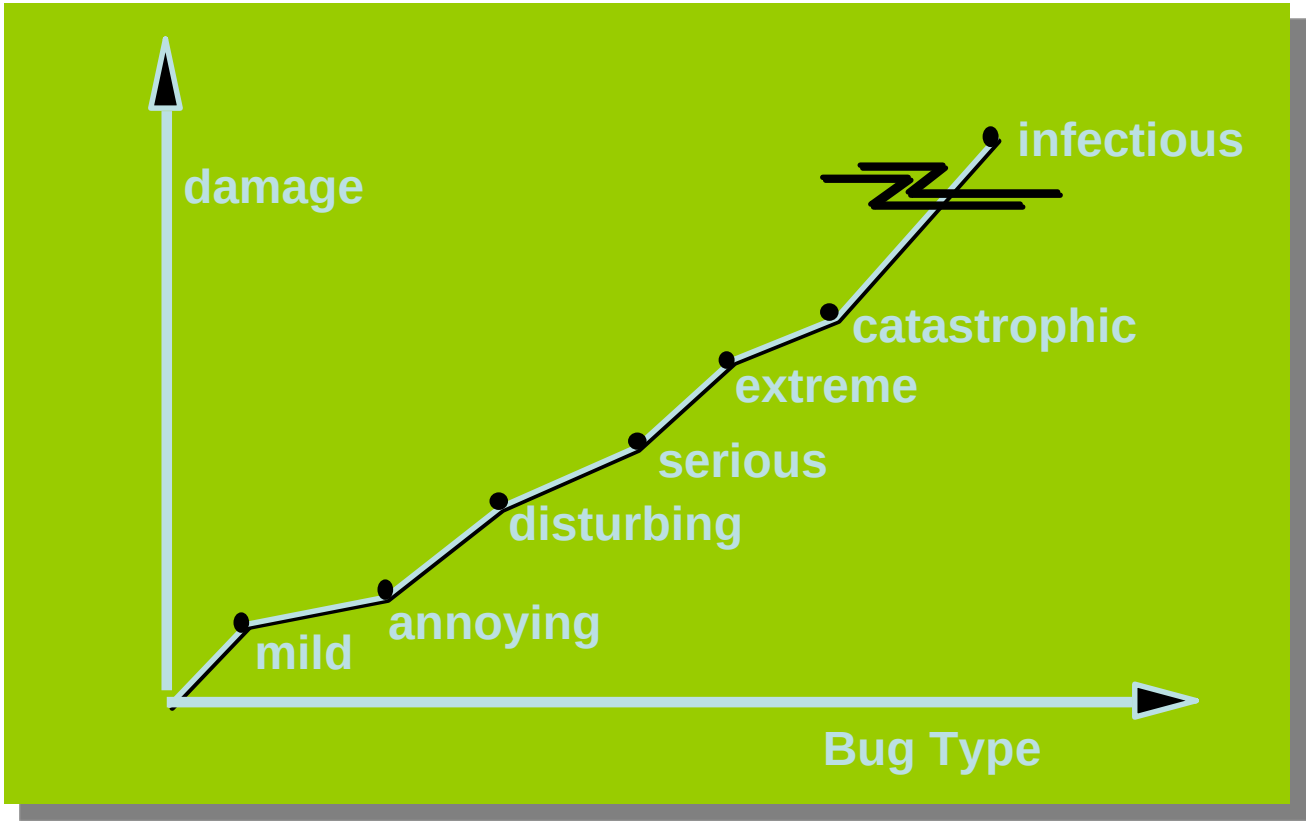


Symptoms & Causes



- symptom and cause may be geographically separated
- symptom may disappear when another problem is fixed
- cause may be due to a combination of non-errors
- cause may be due to a system or compiler error
- cause may be due to assumptions that everyone believes
- symptom may be intermittent

Consequences of Bugs



Bug Categories: **function-related bugs, system-related bugs, data bugs, coding bugs, design bugs, documentation bugs, standards violations, etc.**

Debugging Techniques

- brute force / testing
- backtracking
- induction
- deduction

Correcting the Error

- *Is the cause of the bug reproduced in another part of the program?* In many situations, a program defect is caused by an erroneous pattern of logic that may be reproduced elsewhere.
- *What "next bug" might be introduced by the fix I'm about to make?* Before the correction is made, the source code (or, better, the design) should be evaluated to assess coupling of logic and data structures.
- *What could we have done to prevent this bug in the first place?* This question is the first step toward establishing a statistical software quality assurance approach. If you correct the process as well as the product, the bug will be removed from the current program and may be eliminated from all future programs.

Final Thoughts

- *Think* -- before you act to correct
- Use tools to gain additional insight
- If you're at an impasse, get help from someone else
- Once you correct the bug, use regression testing to uncover any side effects

Q & A