Chapter 13
Software Testing Strategies

Moonzoo Kim

CS Division of EECS Dept.
KAIST
moonzoo@cs.kaist.ac.kr
http://pswlab.kaist.ac.kr/courses/cs550-07

Overview of Ch13. Testing Strategies

- 13.1 A strategic approach to SW testing
- 13.2 Strategic Issues

- 13.3 Test strategies for conventional SW
  - Unit testing
  - Integration testing
- 13.4 Test strategies for OO SW
  - Unit testing
  - Integration testing

- 13.5 Validation testing
- 13.6 System testing

- 13.7 The art of debugging

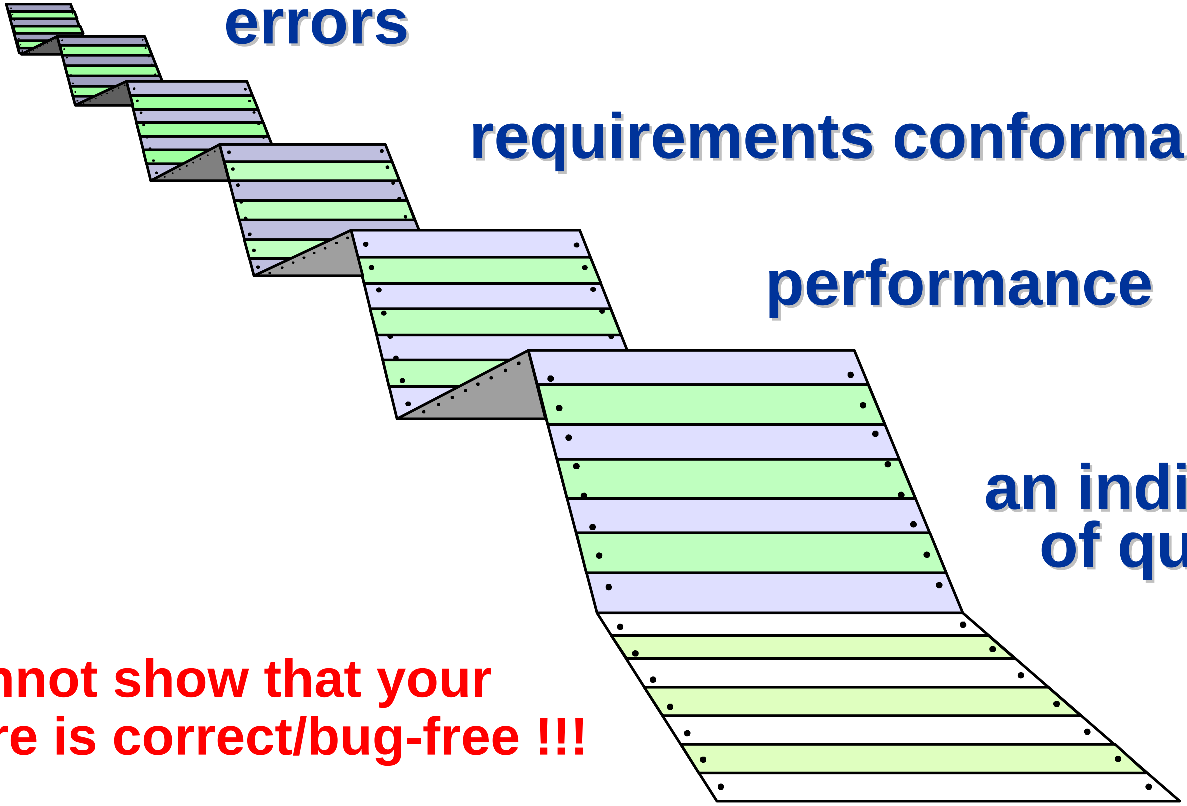
Software Testing

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

Verification: Are we building the product right?

Validation: Are we building the right product?

What Testing Shows



errors

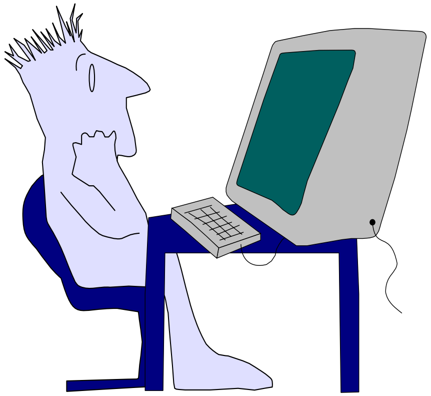
requirements conformance

performance

an indication
of quality

But cannot show that your software is correct/bug-free !!!

Who Tests the Software?

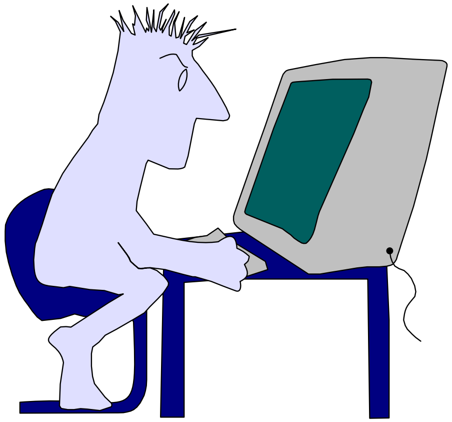


developer

Understands the system

but, will test "gently"

and, is driven by "delivery"



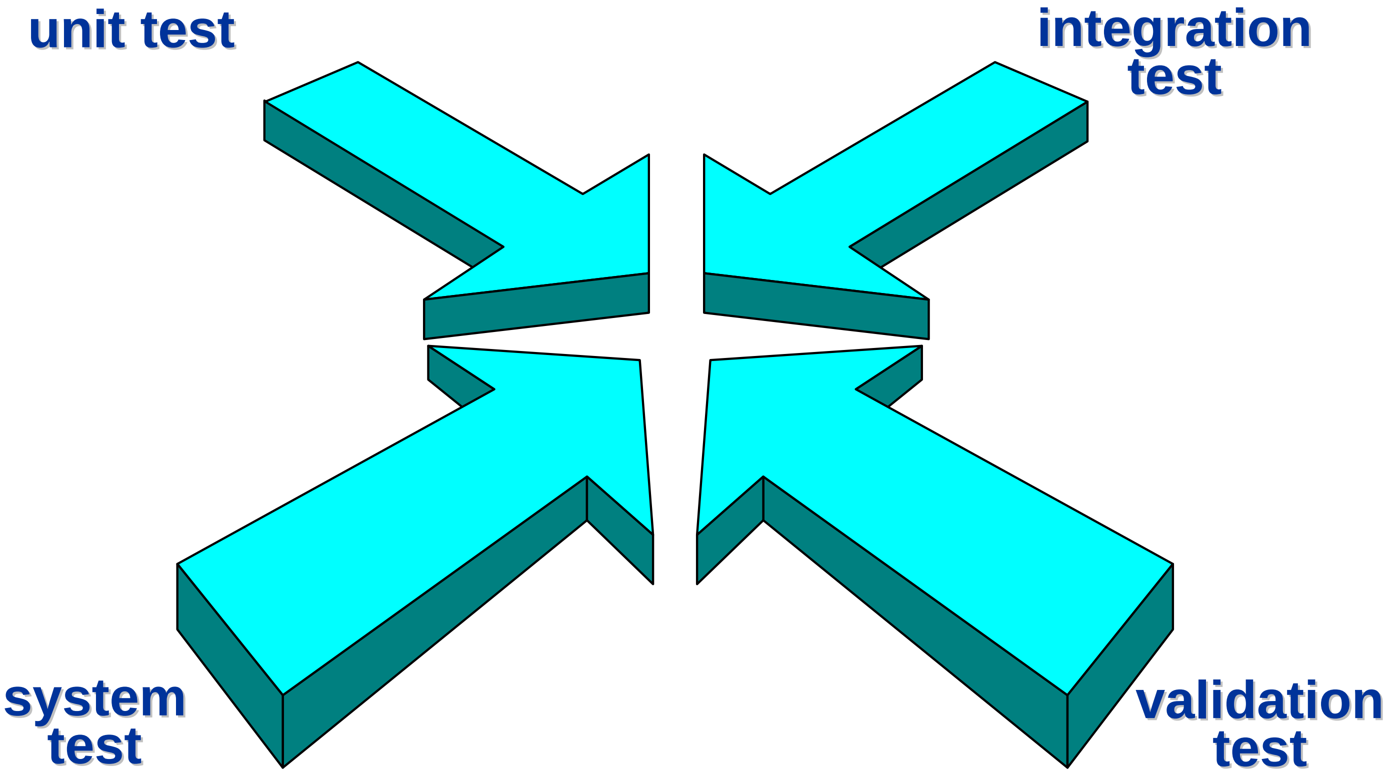
independent tester

Must learn about the system,

but, will attempt to break it

and, is driven by quality

Testing Strategy



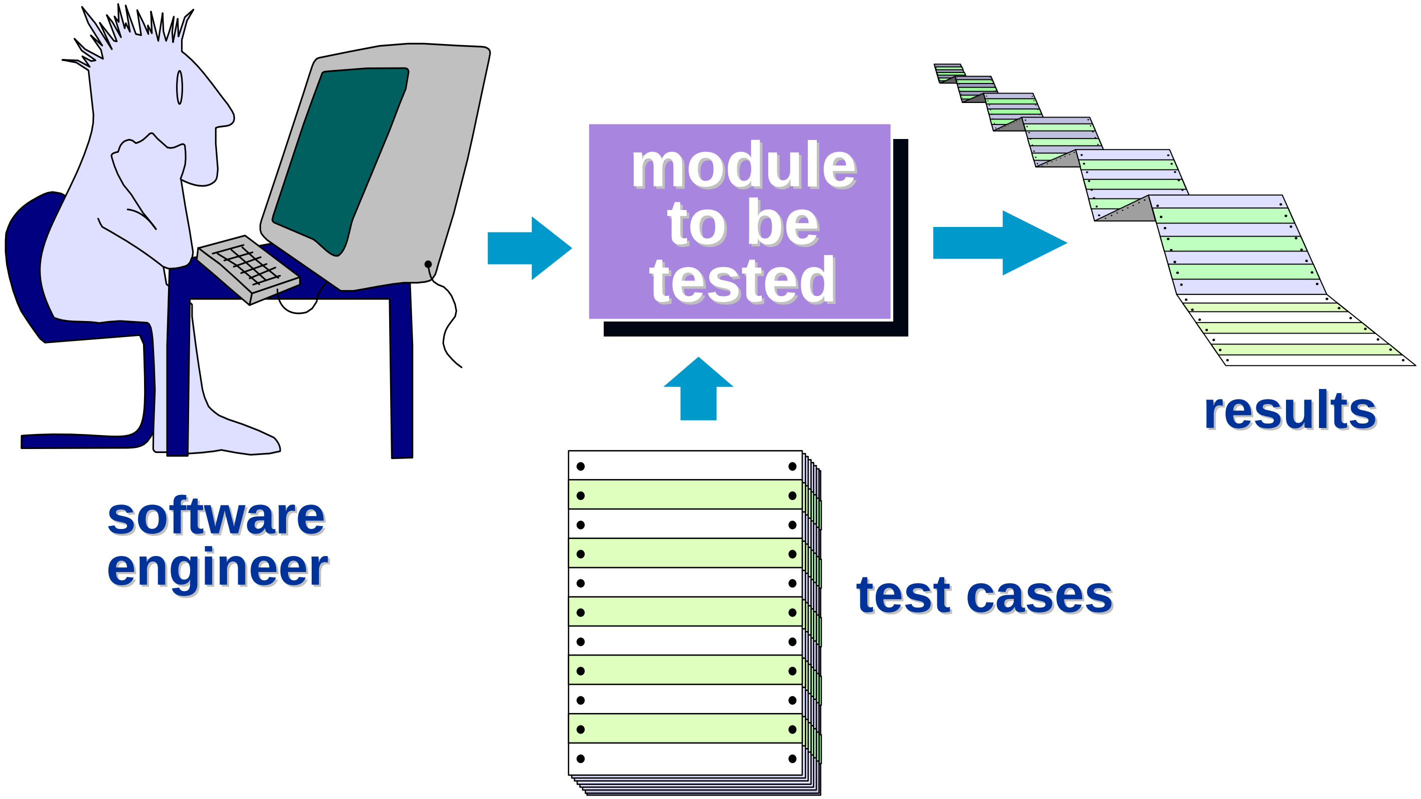
Testing Strategy

- A SW team should conduct effective formal technical reviews. BY doing this, many errors will be eliminated before testing commences
- We begin by ‘testing-in-the-small’ and move toward ‘testing-in-the-large’
- Different testing techniques are appropriate at different points in time
- Testing is conducted be the developer of the SW and an independent test group (ITG)
- Note that testing occur at a time near project deadline. Testing progress must be measurable and problems must surface as early as possible.

Strategic Issues

- 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 formal technical reviews as a filter prior to testing
- Conduct formal technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

Unit Testing

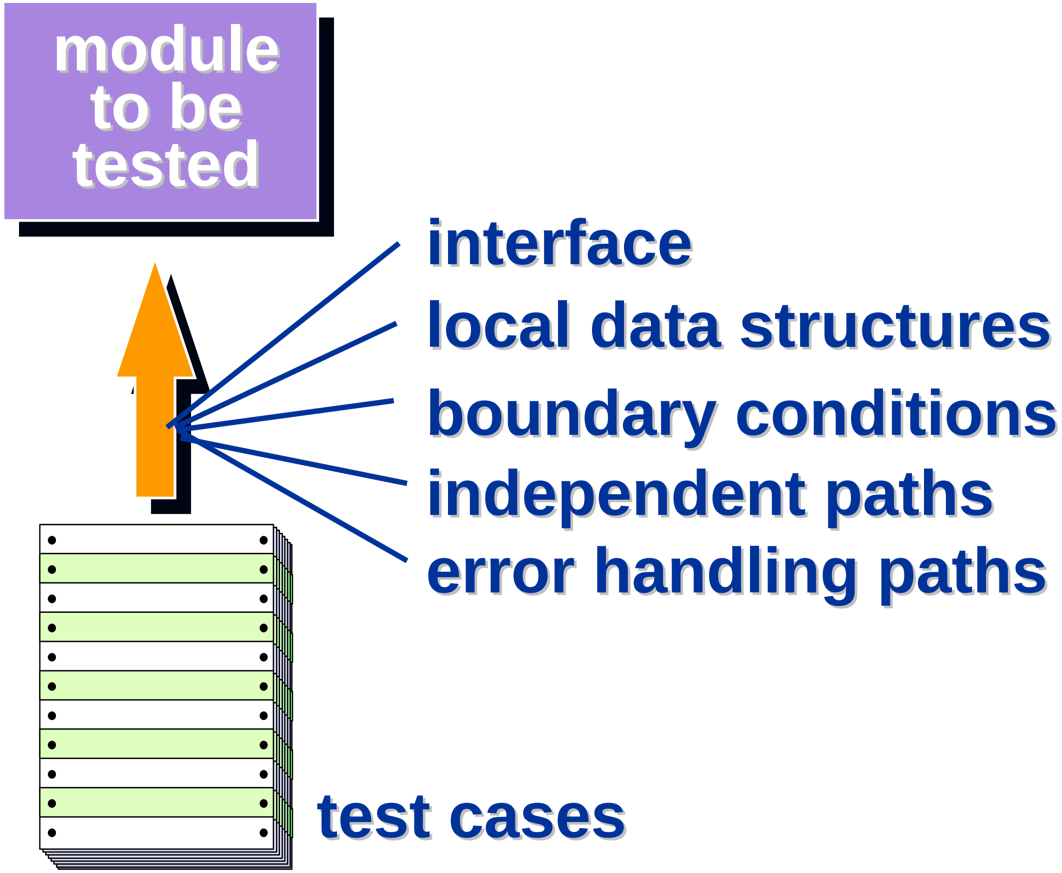

software engineer

module to be tested

results

test cases

Unit Testing



module to be tested

interface

local data structures

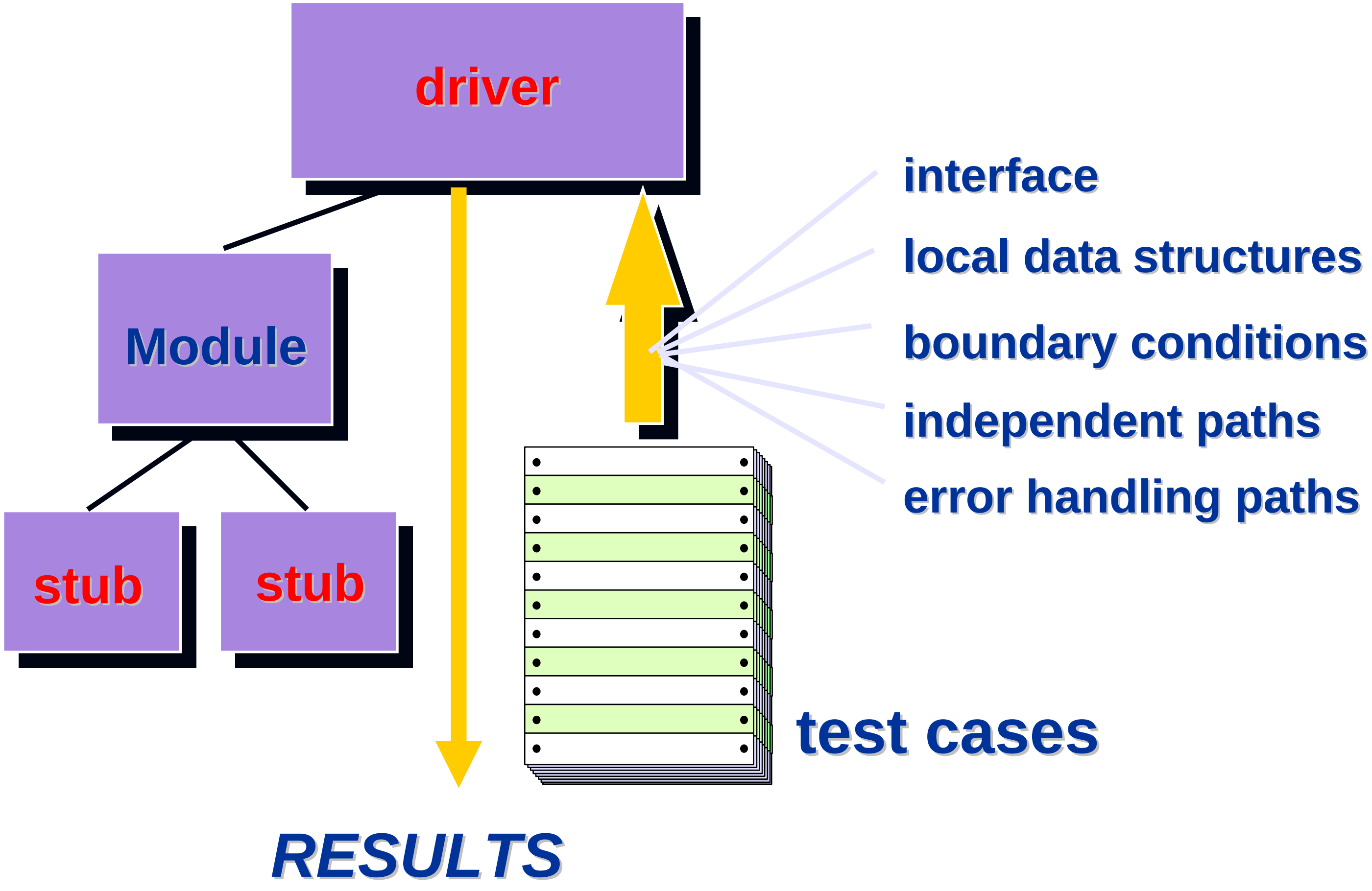
boundary conditions

independent paths

error handling paths

test cases

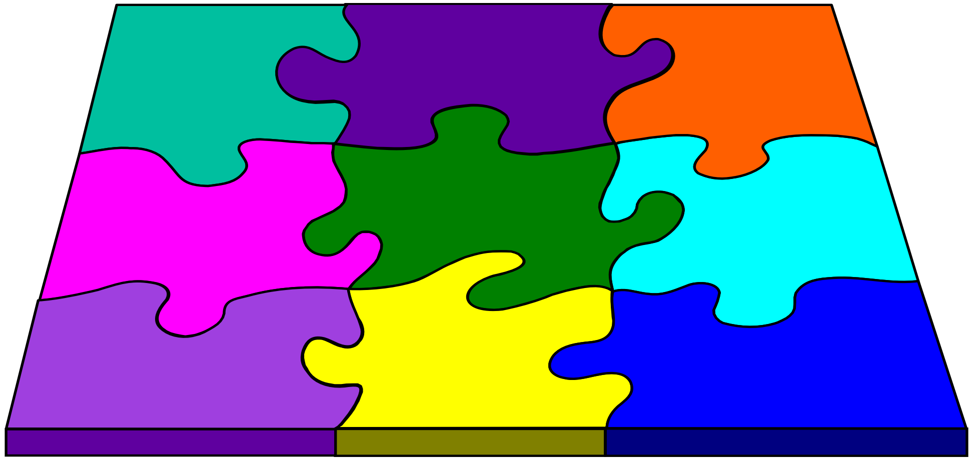
Unit Test Environment



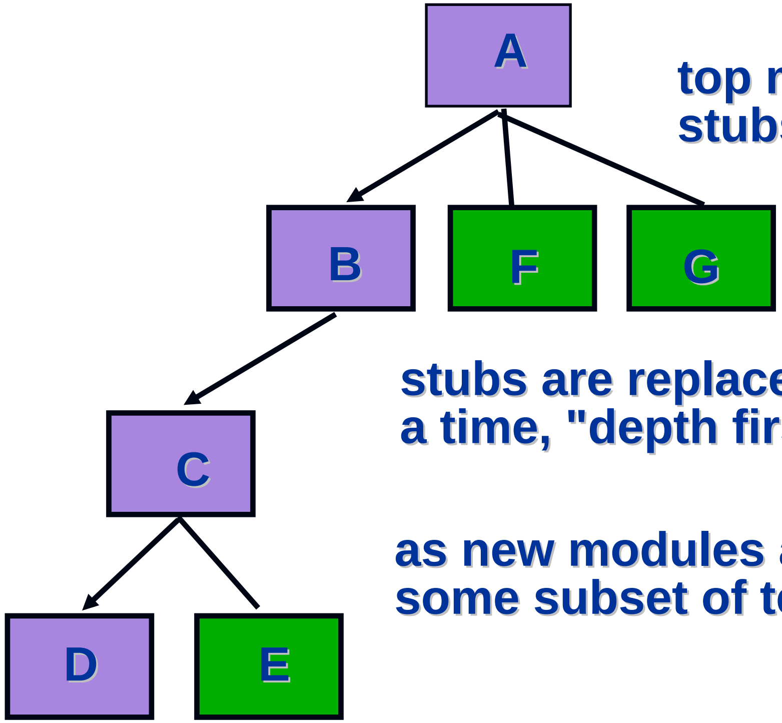
Integration Testing Strategies

Options:

- the “big bang” approach
- an incremental construction strategy



Top Down Integration

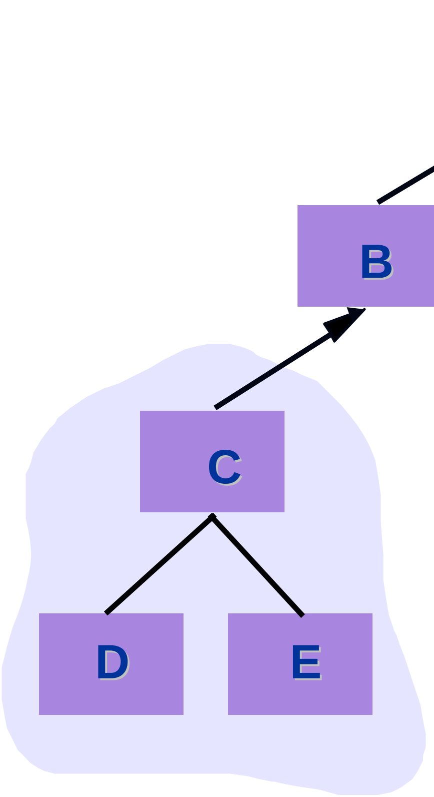


top module is tested with stubs

stubs are replaced one at a time, "depth first"

as new modules are integrated, some subset of tests is re-run

Bottom-Up Integration

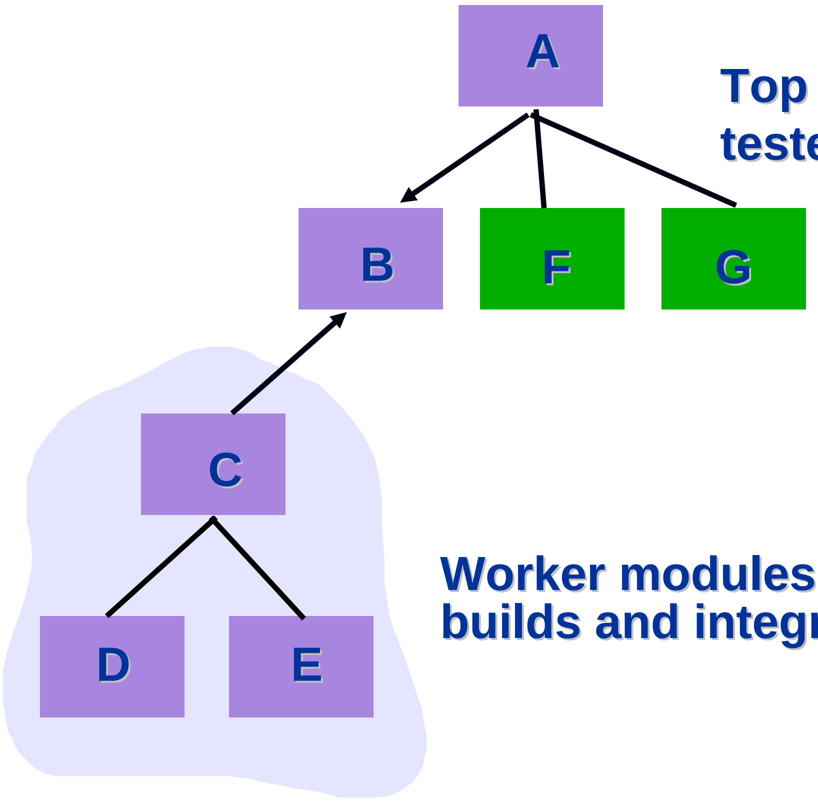


drivers are replaced one at a time, "depth first"

worker modules are grouped into builds and integrated

cluster

Sandwich Testing



Top modules are tested with stubs

Worker modules are grouped into builds and integrated

cluster

Unit Testing in the OO Context

- The concept of the ‘unit’ broadens due to encapsulation
  - A single operation in isolation in the conventional view of unit testing does NOT work
  - Context of a class should be considered
- Comparison
  - Unit testing of conventional SW focus on the algorithmic detail and the data that flow across the module interface
  - Unit testing of OO SW is driven by the operations encapsulated by the class and the state behavior of the class

Example> Effects of Context in OO SW

```
class t {
    int x=10;
    public void f()
      {System.out.println(x));}
}

class t1 extends t {
      public static void
      main(String[] args) {
        (new t1()).f();
    }
}
```

10

```
class t {
    int x=10;
    public void f()
      {System.out.println(x);}
}

class t1 extends t {
    int x=20;
      public static void
      main(String[] args) {
        (new t1()).f();
    }
}
```

10

```
class t {
    int x=10;
    public void f()
      {System.out.println(g());}
    public int g() {return x;}
}

class t1 extends t {
    int x=20;
    public int g() {return x;}
      public static void
      main(String[] args) {
        (new t1()).f();
    }
}
```

20

Integration Testing in the OO Context

- Begins by evaluating the correctness and consistency of the OOA and OOD models
- Testing strategy changes
  - Integration focuses on classes and their execution across a ‘thread’ or in the context of a usage scenario
  - Validation uses conventional black box methods
- Test case design draws on conventional methods, but also encompasses special features
- Test of the CRC Model

OOT Strategy

- Class testing is the equivalent of unit testing
  - operations within the class are tested
  - the state behavior of the class is examined
- Integration applied three different strategies
  - Thread-based testing
    - integrates the set of classes required to respond to one input or event
  - Use-based testing
    - integrates the set of classes required to respond to one use case
  - Cluster testing
    - integrates the set of classes required to demonstrate one collaboration

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.

Preparing for Validation (pg407-408)

- **The scene:**
  - Doug Miller's office, as component-level design continues and construction of certain components begins.
- **The players:**
  - Doug Miller

    software engineering manager,
  - Vinod, Jamie, Ed, Shakira

    members of the *SafeHome* software engineering team.
- **The conversation:**
- **Doug:** The first increment will be ready for validation in what ... about three weeks?
- **Vinod:** That's about right. Integration is going well. We're smoke testing daily, finding some bugs but nothing we can't handle. So far, so good.
- **Doug:** Talk to me about validation.
- **Shakira:** Well, we'll use all of the use-cases as the basis for our test design. I haven't started yet, but I'll be developing tests for all of the use-cases that I've been responsible for.
- **Ed:** Same here.
- **Jamie:** Me too, but we've got to get our act together for

acceptance testing and also for alpha and beta testing, no?

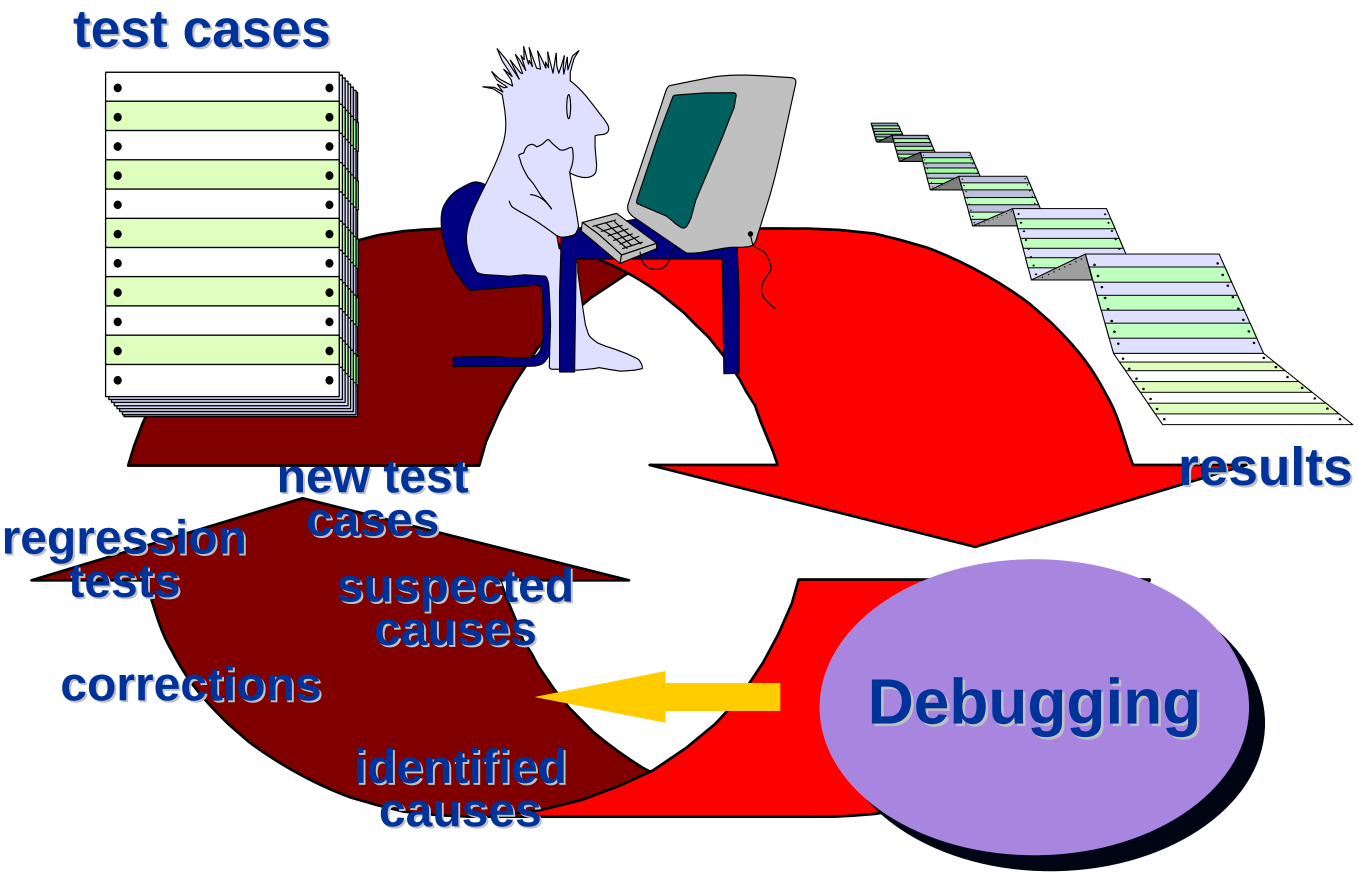
- **Doug:** Yes, In fact I've been thinking that we could bring in an outside contractor to help us with validation. I have the money in the budget ... and it would give us a new point of view.
- **Vinod:** I think we've got it under control.
- **Doug:** I'm sure you do, but an ITG gives us an independent look at the software.
- **Jamie:** We're tight on time here, Doug. I, for one, don't have the time to baby-sit anybody you bring in to do the job.
- **Doug:** I know, I know. But if an ITG works from requirements and use-cases, not too much baby sitting will be required.
- **Vinod:** I still think we've got it under control.
- **Doug:** I hear you, Vinod, but I'm going to overrule on this one. Let's plan to meet with the ITG rep later this week. Get 'em started and see what they come up with.
- **Vinod:** Okay, maybe it'll lighten the load a bit.

High Order Testing

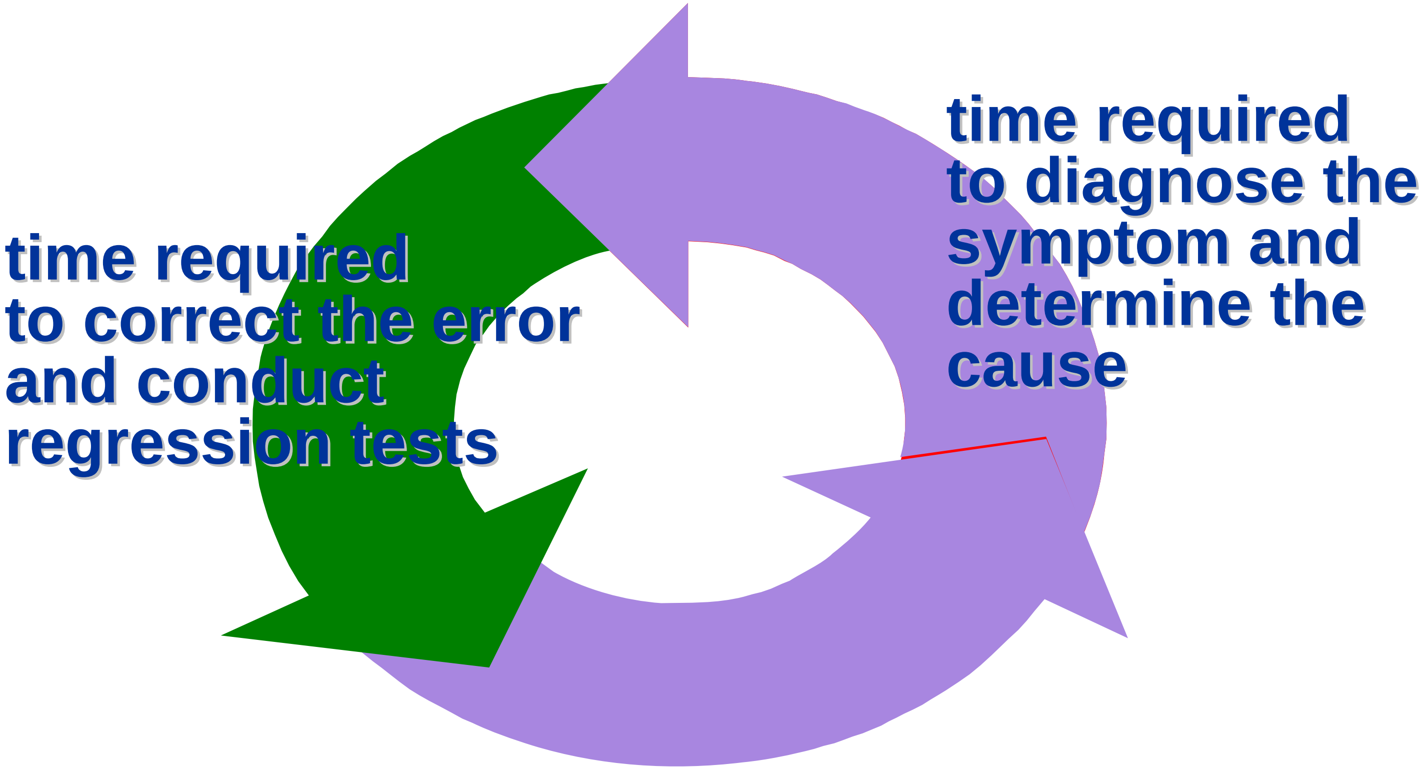
- Validation testing
  - Focus is on software requirements
- System testing
  - Focus is on system integration
- Alpha/Beta testing
  - Focus is on customer usage
- Recovery testing
  - forces the software to fail in a variety of ways and verifies that recovery is properly performed
- Security testing
  - 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



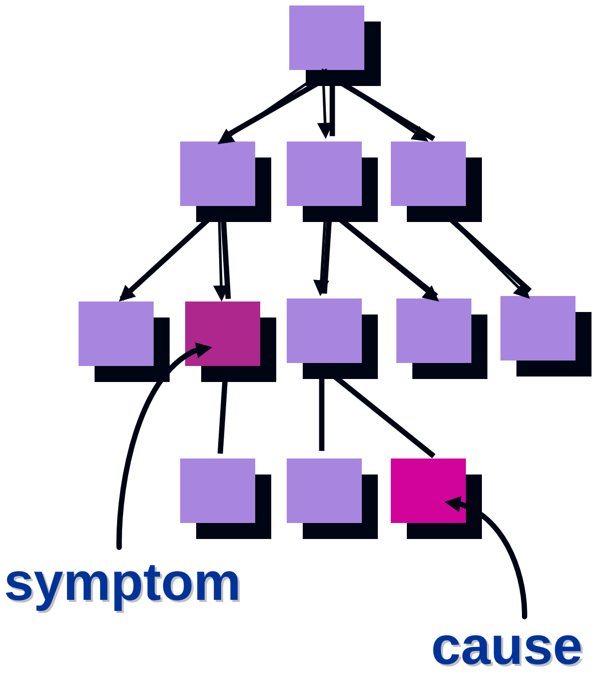
Debugging Effort



time required
to correct the error
and conduct
regression tests

time required
to diagnose the
symptom and
determine the
cause

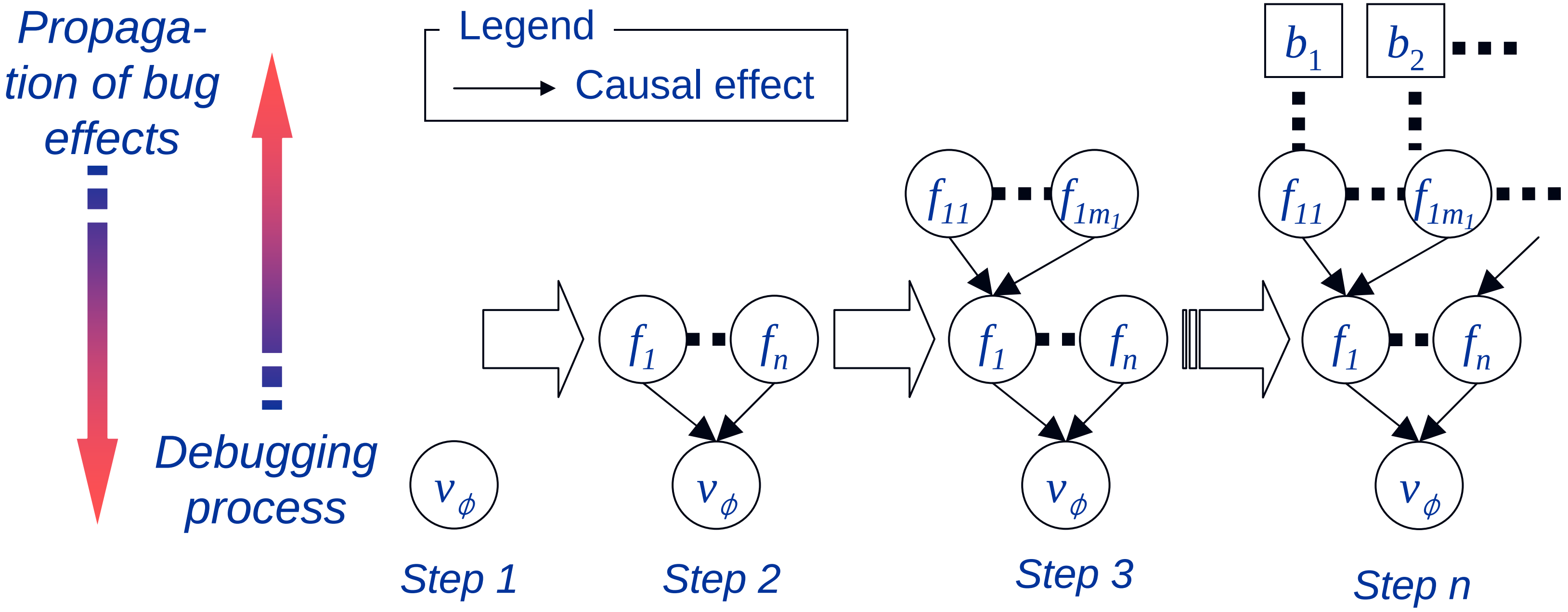
Symptoms & Causes



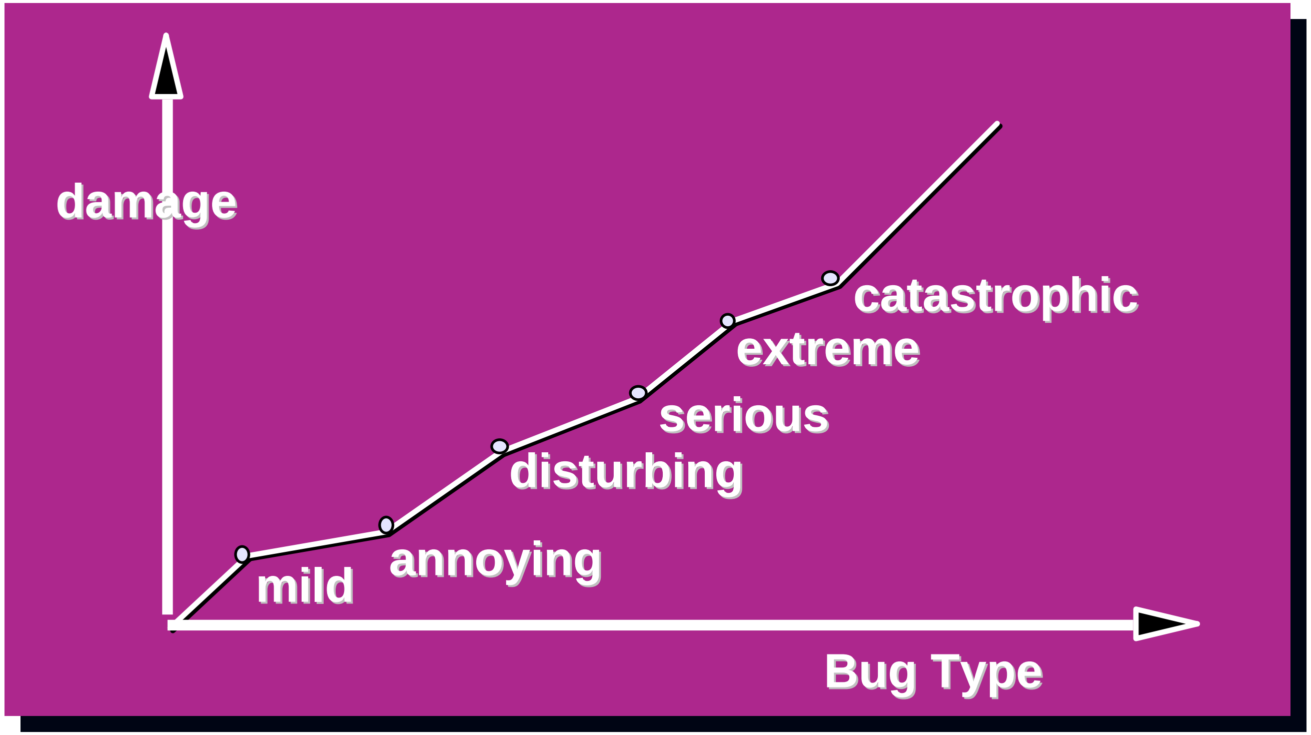
- ❑ symptom and cause may be geographically separated
- ❑ symptom may disappear when another problem is fixed
- ❑ cause may be due to a combination of non-errors
- ❑ cause may be due to a system or compiler error
- ❑ cause may be due to assumptions that everyone believes
- ❑ symptom may be intermittent

Failures v.s. faults *IEEE standard 610.12*

Symptoms & Causes (cont.)



Consequences of Bugs



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

Debugging: Final Thoughts

1. Don't run off in haste, <u>think</u> about the symptom you're seeing.

2. <u>Use tools</u> (e.g., dynamic debugger) to gain more insight.

3. If at an impasse, <u>get help</u> from someone else.

4. Be absolutely sure to <u>conduct regression tests</u> when you do "fix" the bug.