SOFTWARE ENGINEERING

CHAPTER-14
SOFTWARE TESTING STRATEGIES

Software Engineering: A Practitioner's Approach, 7th edition

Originated by Roger S. Pressman

Software School, Fudan University
Spring Semester, 2016

Software Testing

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

Test Case

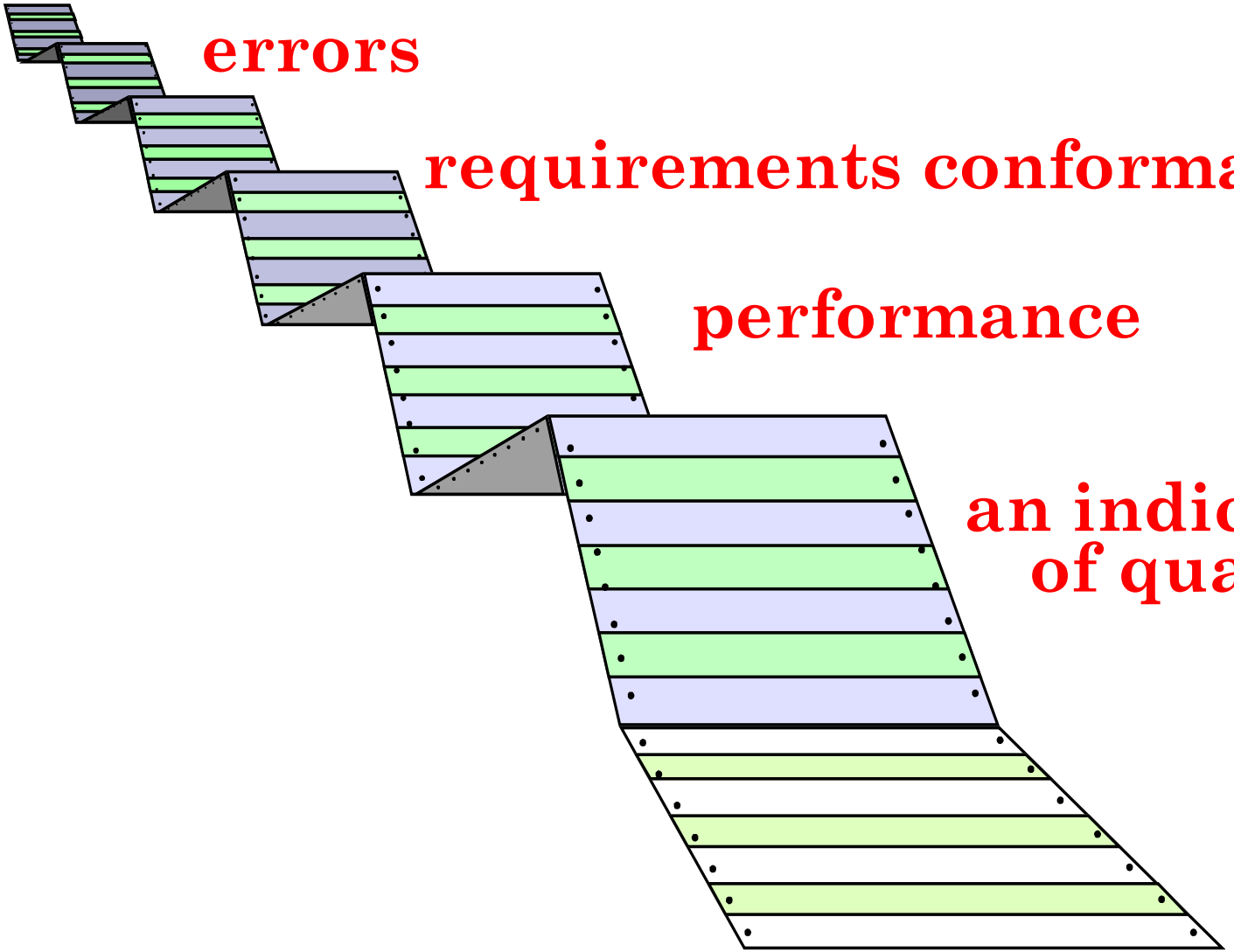
- Test Case: a set of conditions or variables (input) under which a tester will determine whether an application or software system is working correctly or not (expected output)
  - Often referred to as test scripts, particularly when written
  - It may take many test cases to determine that a software program or system is likely to be correct

TEST ORACLE

from wikipedia

- The mechanism for determining whether a software program or system has passed or failed (determining the expected output)
- It is used by comparing the output(s) of the system under test, for a given test case input, to the outputs that the oracle determines that product should have
- Oracles are always separate from the system under test, and usually can be:
  - specifications and documentation
  - other products
  - an heuristic oracle that provides approximate results or exact results
  - a statistical oracle that uses statistical characteristics
  - or a human being's judgment
  - ……

What Testing Shows



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.

V&V: Verification and Validation

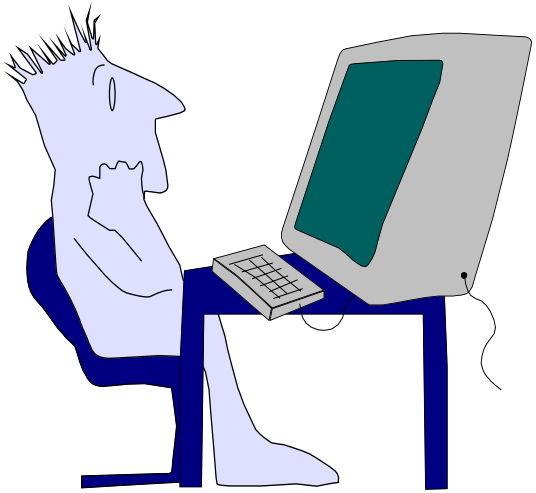
a strong divergence about what types of testing constitute "validation"

- Verification (验证)
  - Are we building the product right?
  - Ensure that software correctly implements a specific function
- Validation (确认)
  - Are we building the right product?
  - Ensure that software built is traceable to customer requirements
- Testing and V&V
  - Testing plays an extremely important role in V&V
  - But many other activities are also necessary, e.g. review, simulation, monitoring

Some "General Knowledge" about Software Testing

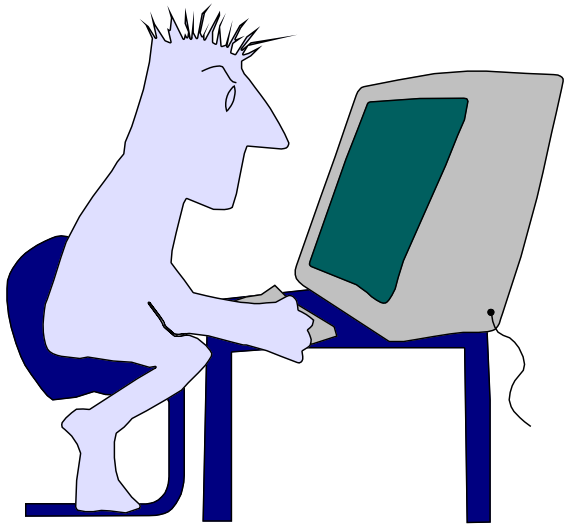
- Testing is a sampling method to find errors and measure software quality
- Testing is costly and time-consuming
- Theoretically, testing can never be complete
- More testing is always better, but we can not wait too long (timely delivery and cost issue)
- Although far from perfect, we can do better by following some good practices

WHO TESTS THE SOFTWARE?



developer

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



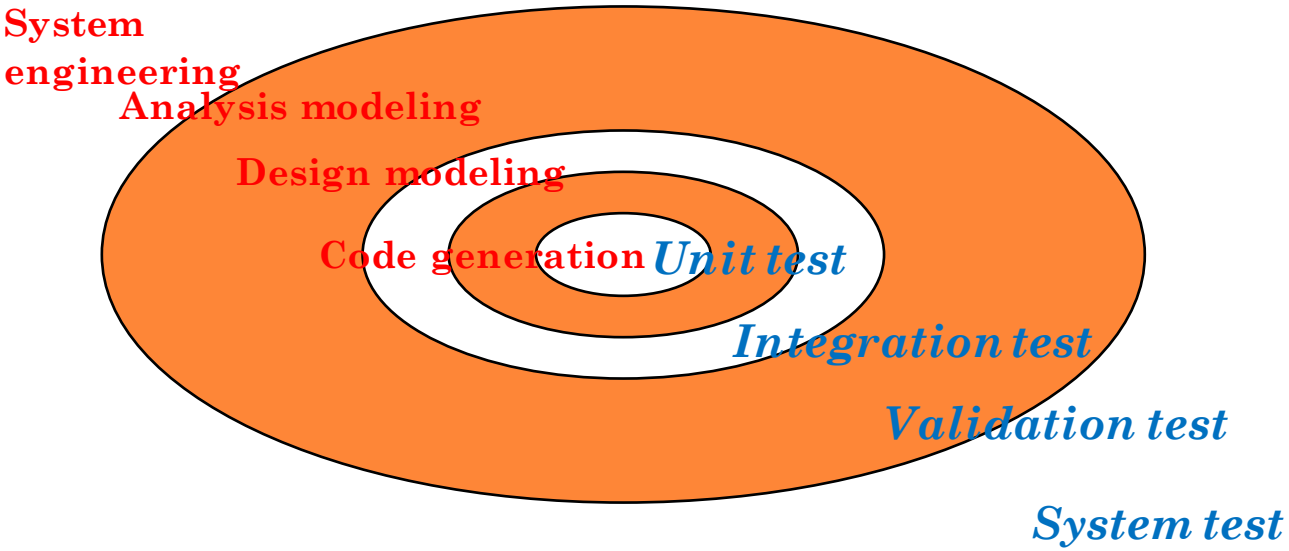
independent tester

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

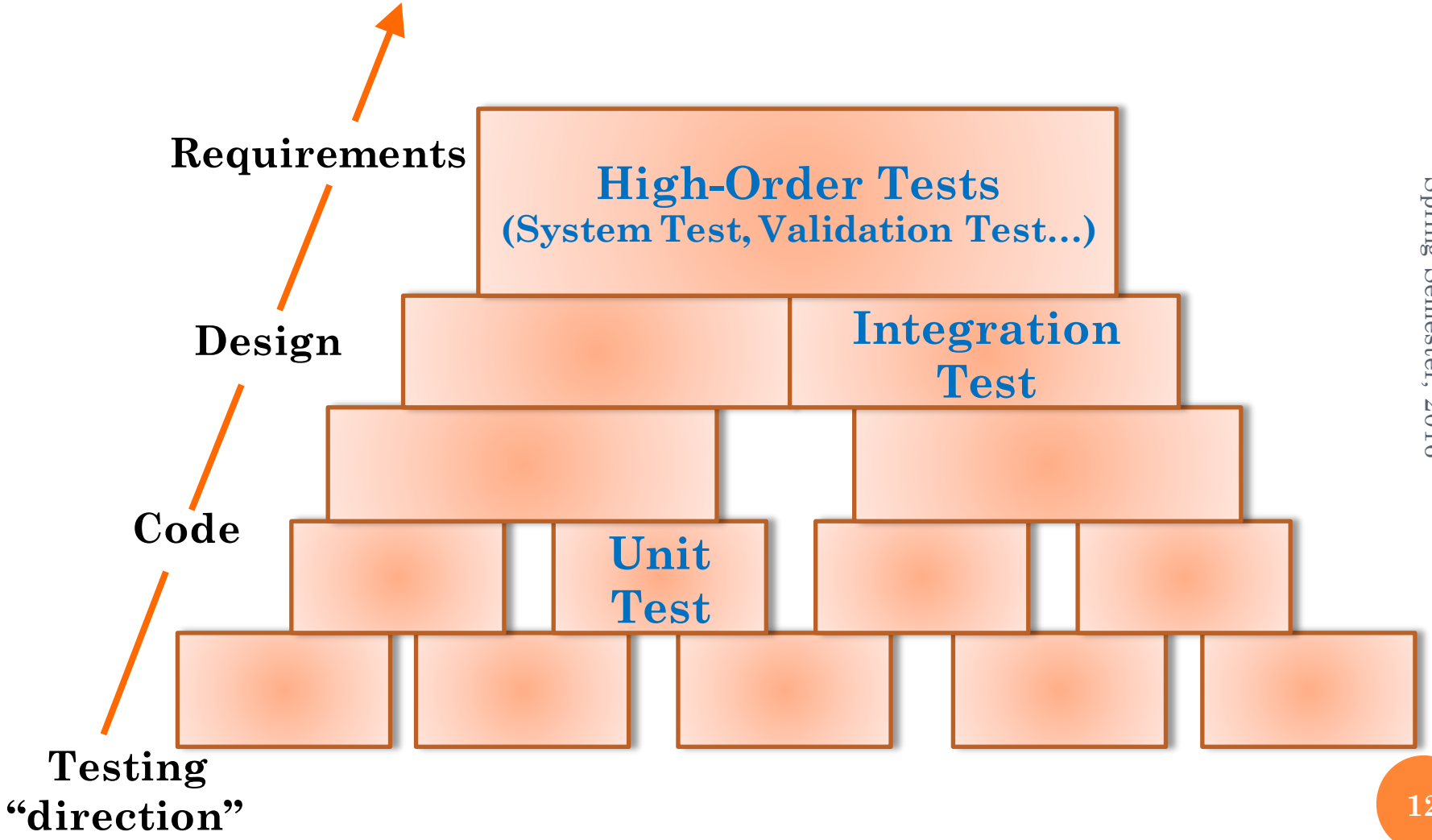
Testing Strategy: from Small to Large

- 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

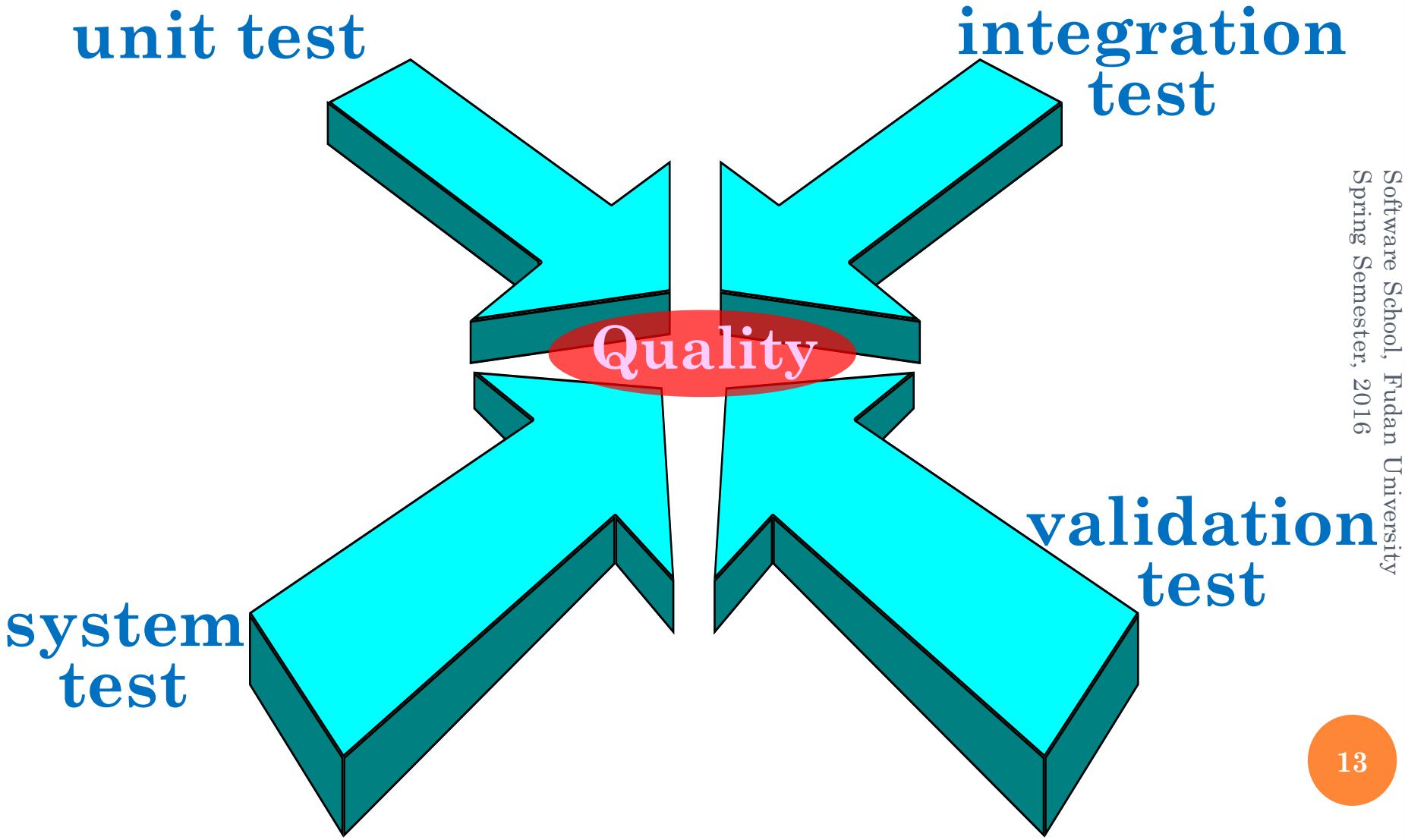
TESTING STRATEGY



Software Testing Steps



Testing Strategy



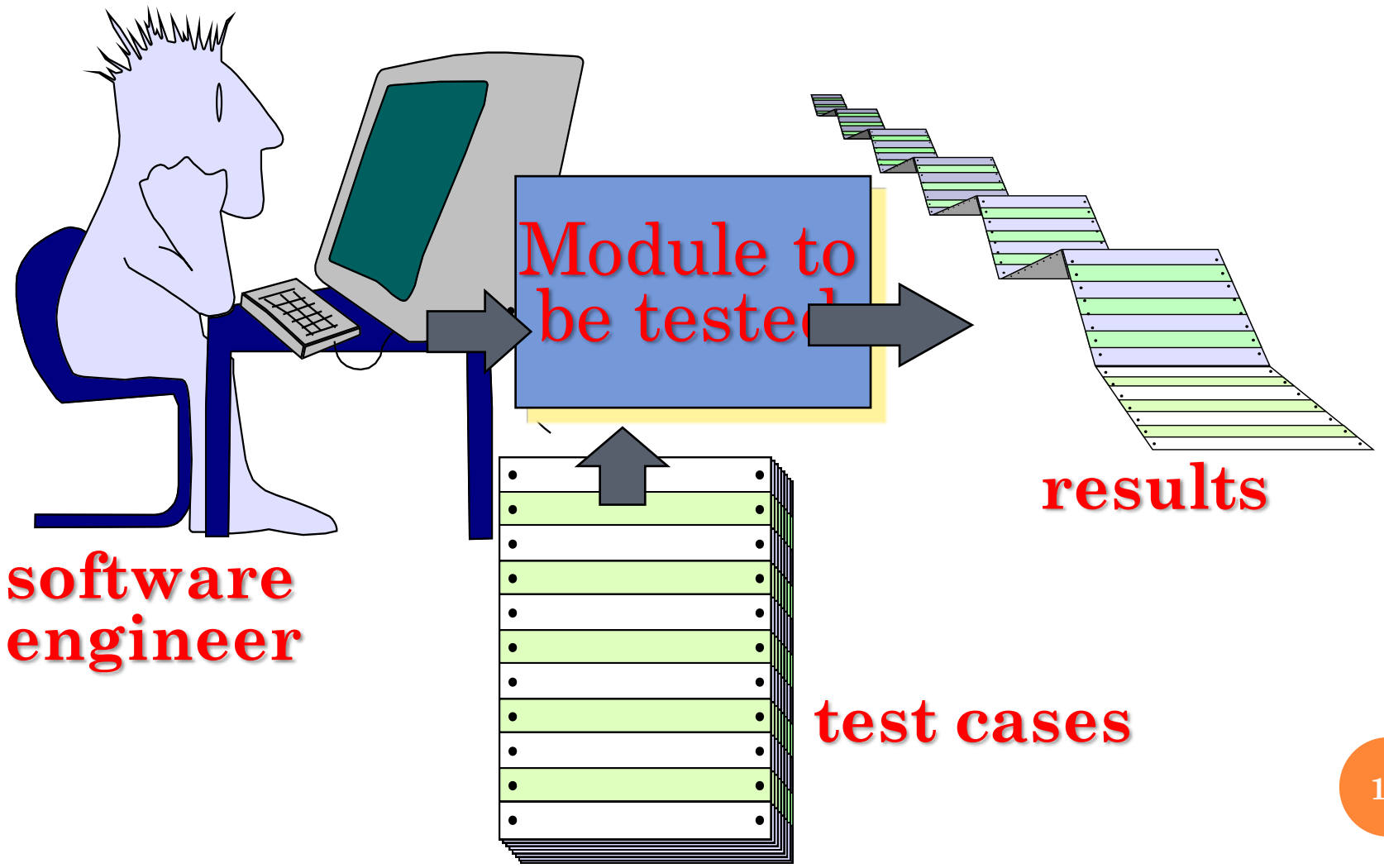
Strategic Issues

- 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*"

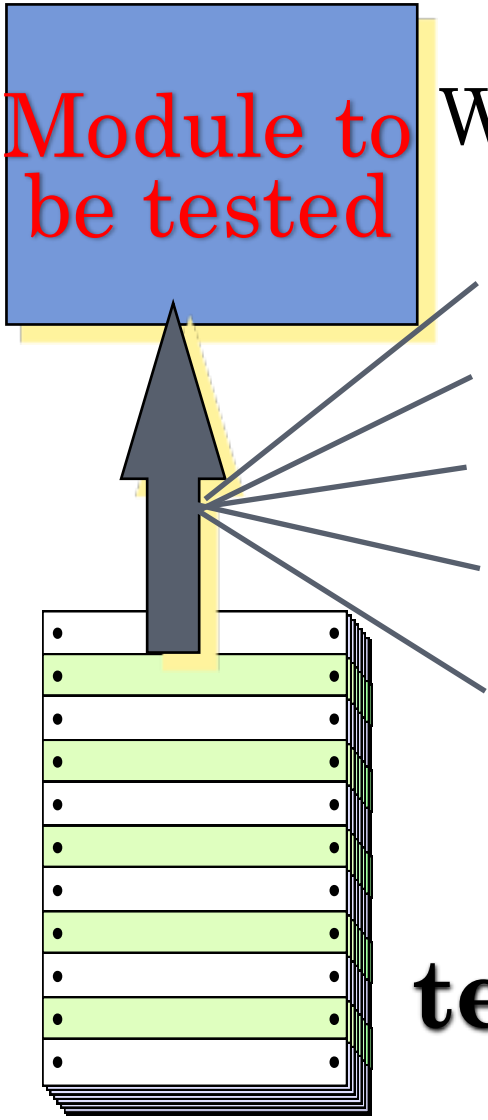
Strategic Issues (Cont.)

- 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



UNIT TESTING (CONT.)

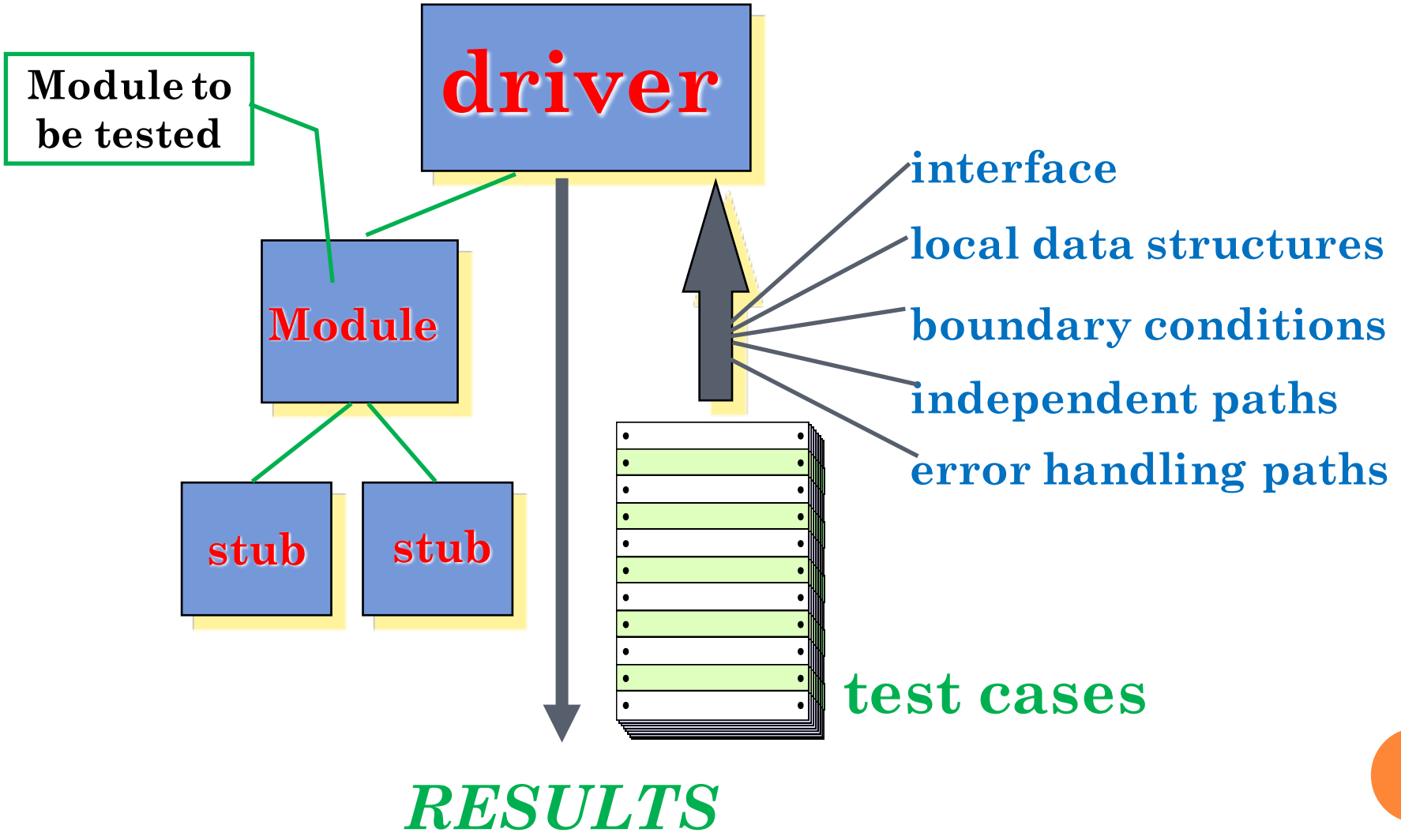


What to test?

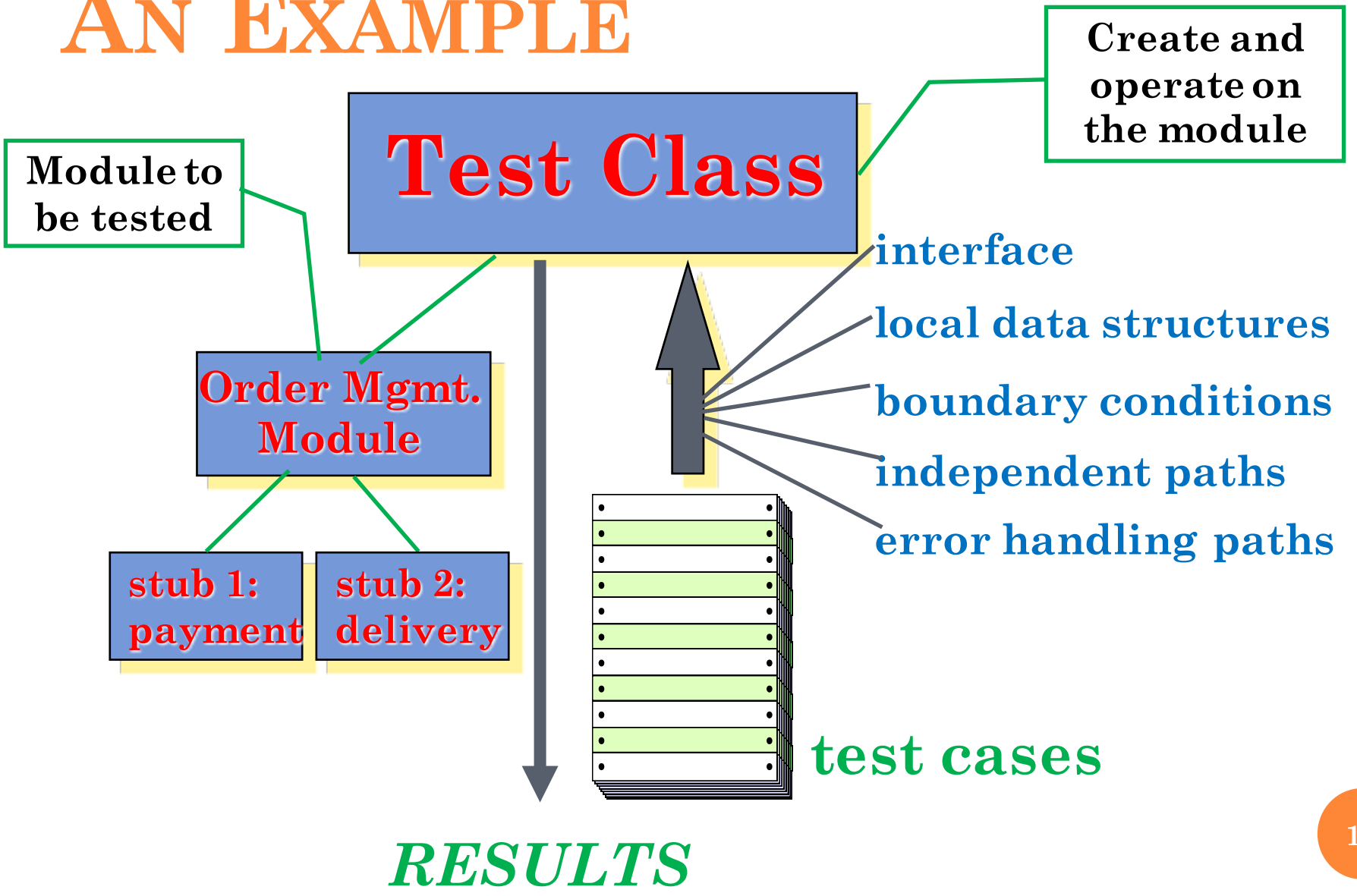
- **interface**
- **local data structures**
- **boundary conditions**
- **independent paths**
- **error handling paths**

test cases

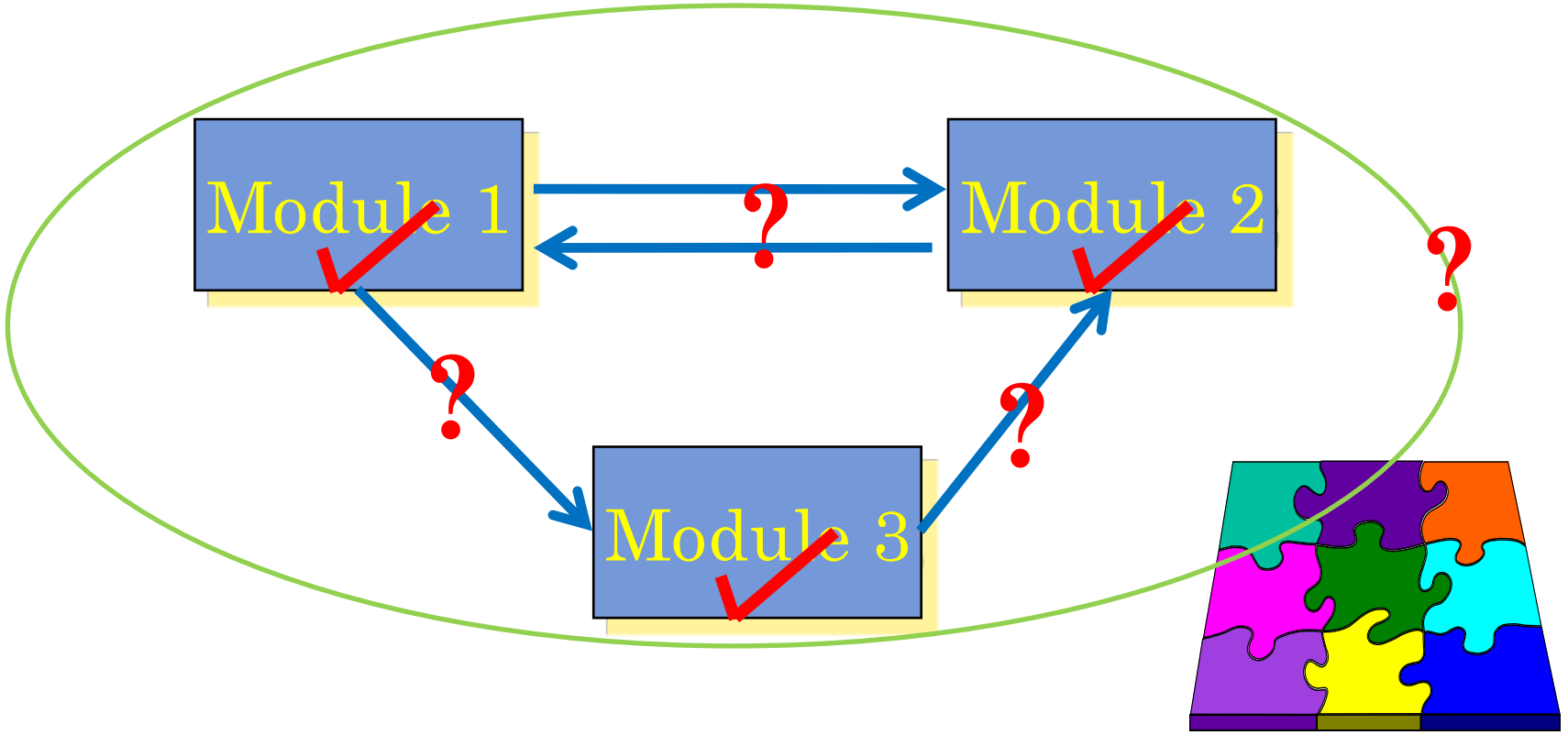
UNIT TEST ENVIRONMENT



Unit Test Environment: An Example

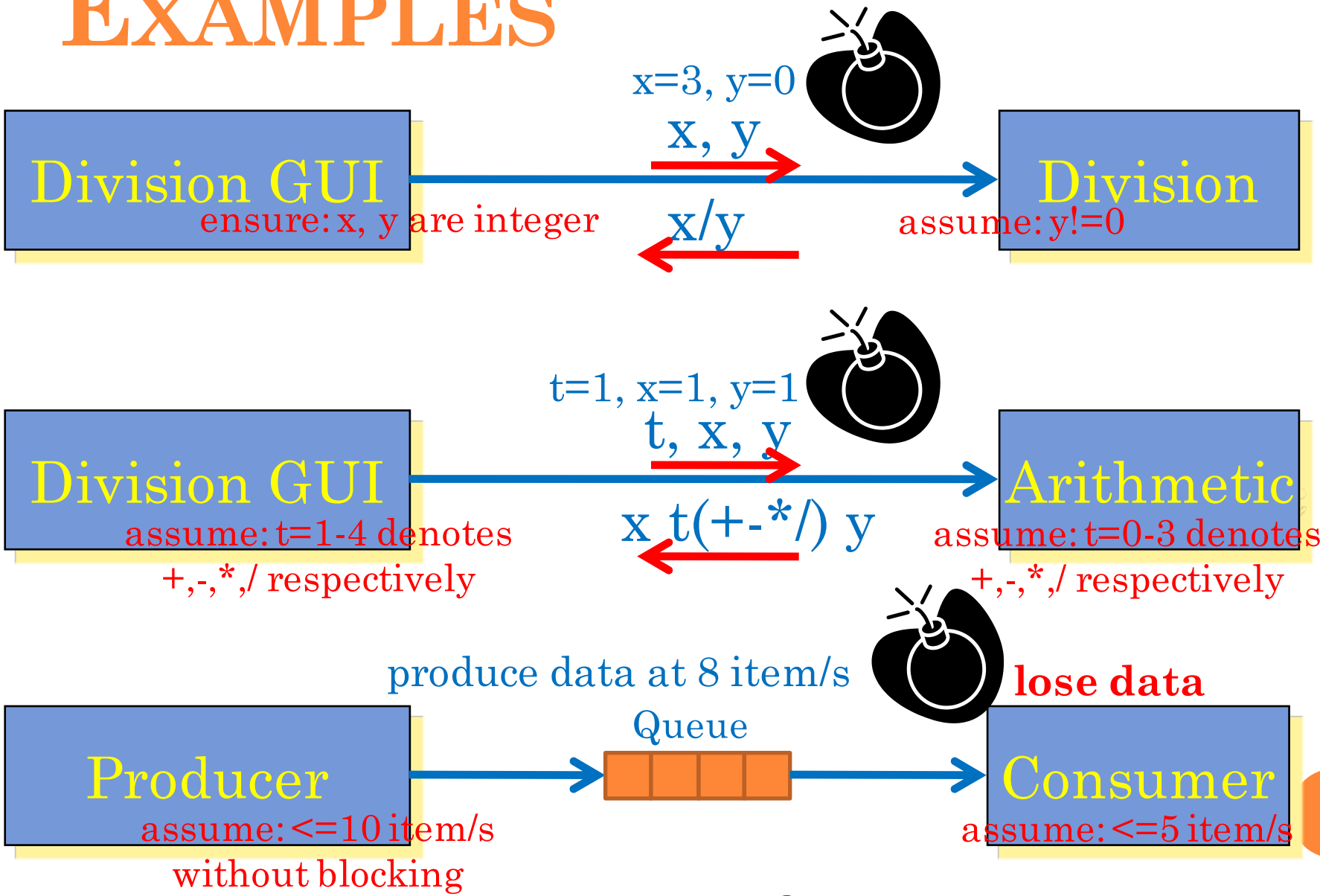


Integration Testing



- Integration testing: a systematic technique for constructing the software architecture while at the same time conducting tests to uncover errors associated with interfacing
- Objective: take unit tested components and build a program structure that has been dictated by design

Integration Error: Examples



Inconsistent interface assumption

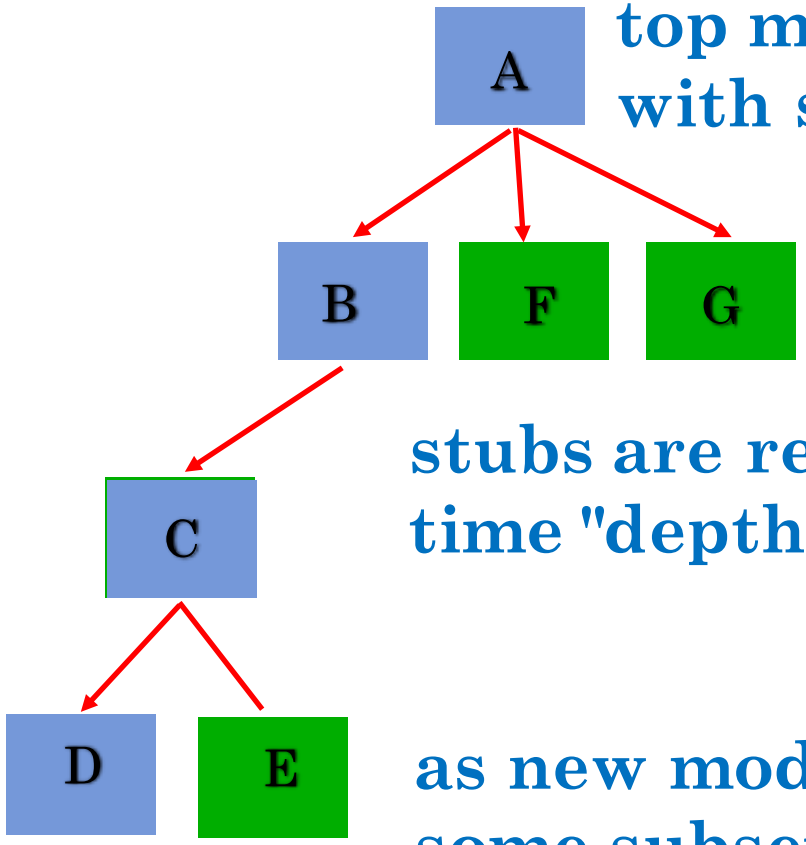
Integration Error: Other Examples

- One module can have an inadvertent, adverse affect on another
- Individually acceptable imprecision may be magnified to unacceptable levels
  - E.g. a salary calculation module with an error of 0.01 causes unacceptable errors when it is integrated to compute salary for more than 10,000 employees
- Global data structures present problems
  - Global data can be regarded as another kind of "interface" among different modules
- Shared resources present problems
  - E.g. shared critical resources: competition, dead lock…

Integration Testing Options

- The "big bang" approach
  - All components are combined in advance, then the entire program is tested as a whole
  - Problems: chaos, and a lot of errors; correction is difficult because isolation of causes is complicated by the vast expanse of the entire program
- Incremental construction strategy
  - Modules are constructed and tested in small increments
  - Merits: Errors are easier to isolated and correct; interfaces are more likely to be tested completely; systematic test approach may be applied

Top Down Integration: Depth First

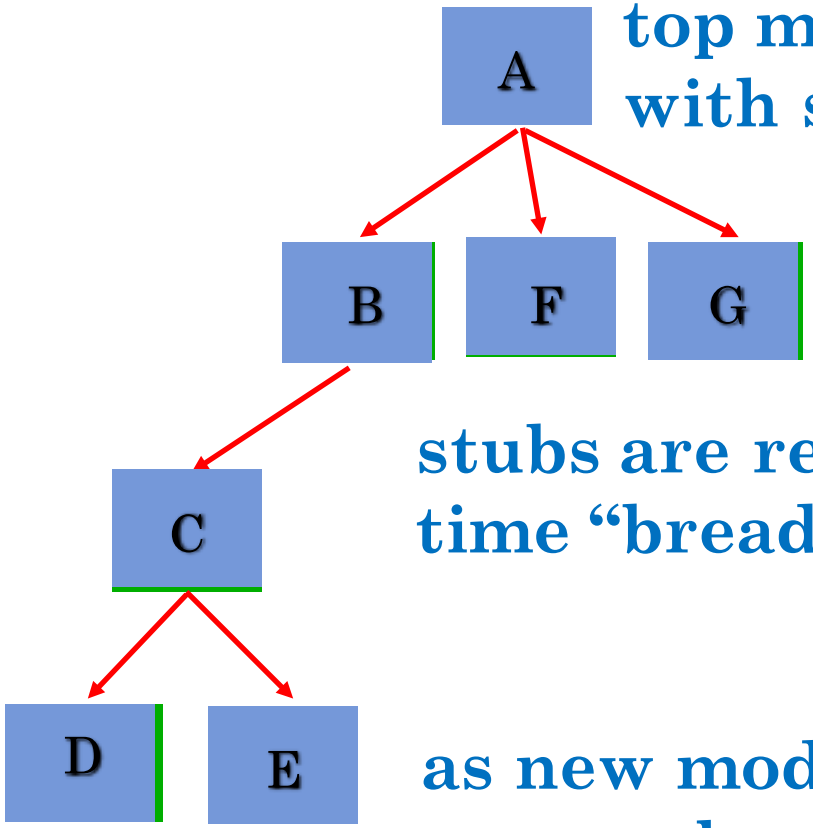


top module is tested with stubs

stubs are replaced one at a time "depth first"

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

Top Down Integration: breadth First

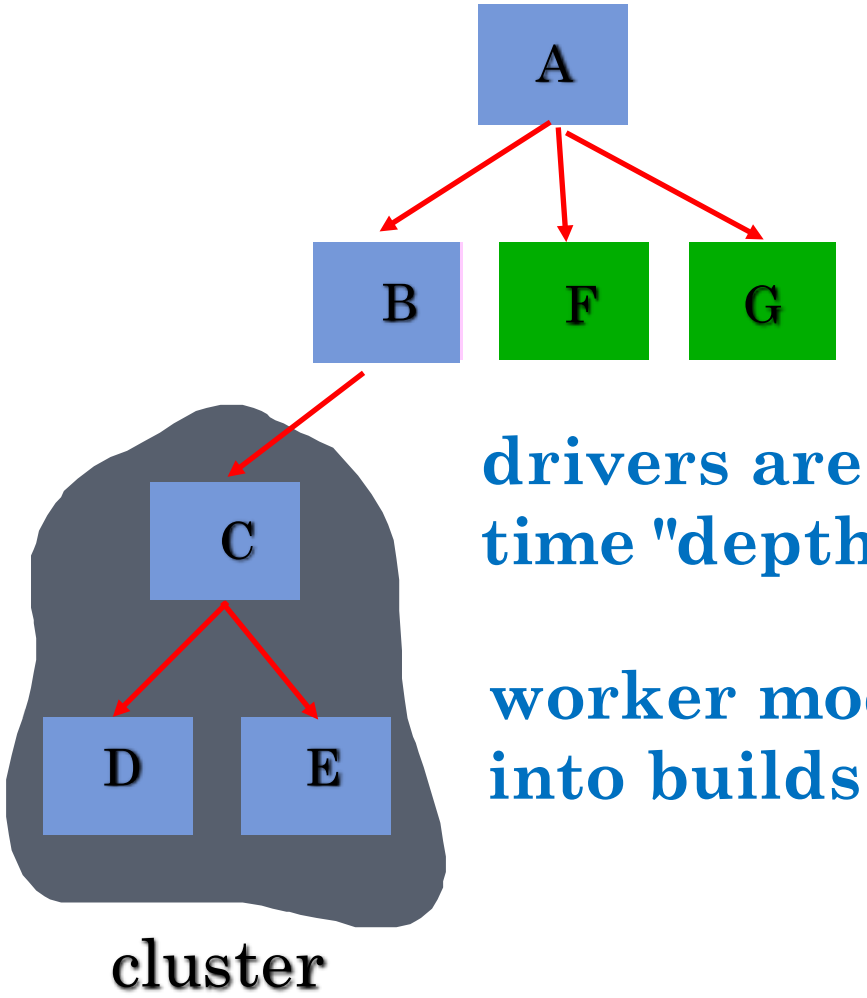


top module is tested with stubs

stubs are replaced one at a time “breadth first"

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

Bottom-Up Integration

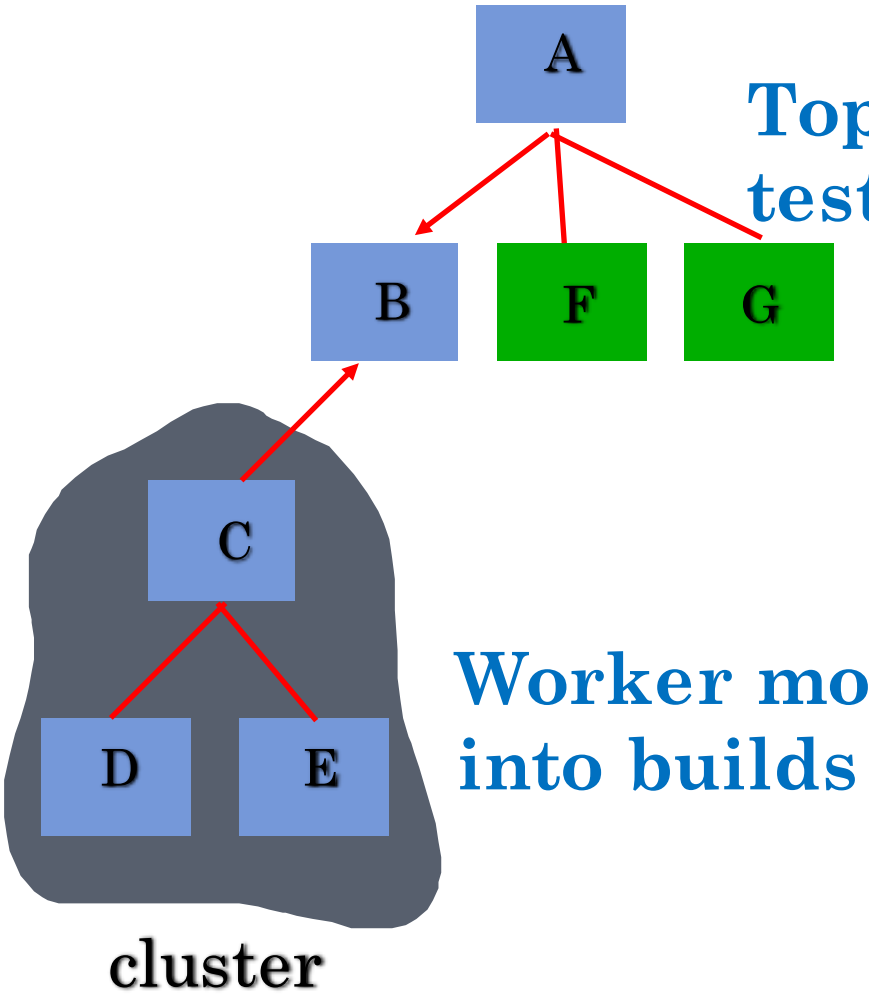


drivers are replaced one at a time "depth first"

worker modules are grouped into builds and integrated

SANDWICH TESTING

Combined Top-down and Bottom-up Integration



Top modules are tested with stubs

Worker modules are grouped into builds and integrated

OBJECT-ORIENTED TESTING

- Begins by evaluating the correctness and consistency of the OOA and OOD ***models***
  - Isn't testing related to coding only??
- Testing strategy changes
  - the concept of the 'unit' broadens due to *encapsulation*
  - 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

OO Testing Strategy

- Class testing is the equivalent of unit testing:
  - ***operations*** within the class are tested
  - the ***state behavior*** of the class is examined
- Integration applies 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

BROADENING THE VIEW OF "TESTING"

Testing in OO analysis and design

It can be argued that the review of OO analysis and design models is especially useful because the same semantic constructs (e.g., classes, attributes, operations, messages) appear at the analysis, design, and code level.

Therefore, a problem in the definition of class attributes that is uncovered during analysis will circumvent *side effects* that might occur if the problem were not discovered until design or code (or even the next iteration of analysis).

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

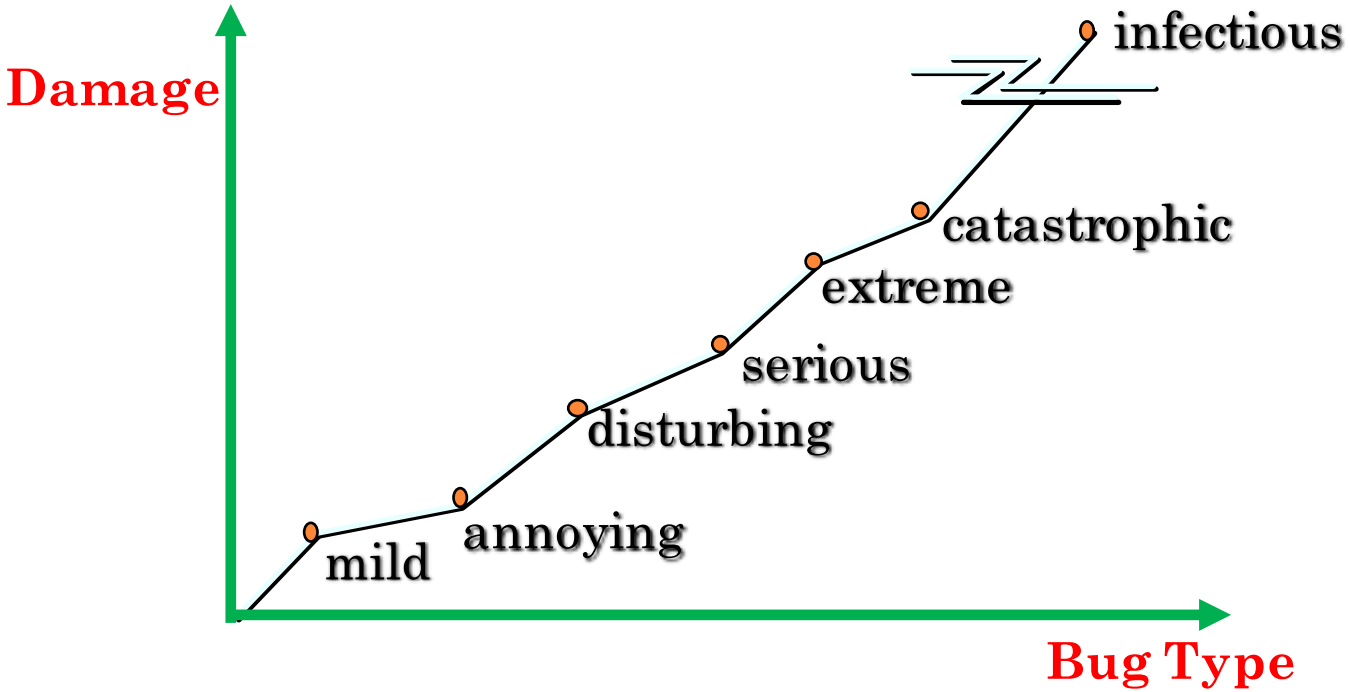
High Order Testing

- **Validation Testing:** focus on requirements
  - **Configuration review (Audit):** to ensure that all elements of the software configuration have been properly developed and cataloged with necessary details for supporting
  - **Acceptance test:** for the customer to validate all requirements, usually conducted by end-users
  - **Alpha/Beta Testing:** focus on customer usage
    - **Alpha Testing:** conducted at the developer's site by end-users and in a controlled environment; the developer observes the usage process and record errors and problems
    - **Beta Testing:** conducted at end-user sites and the developer is generally not present; a "live" application of the software that cannot be controlled by the developer; the end-user records problems and reports them to the developer at regular intervals

High Order Testing (Cont.)

- **System Testing:** focus on system integration beyond software
  - **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

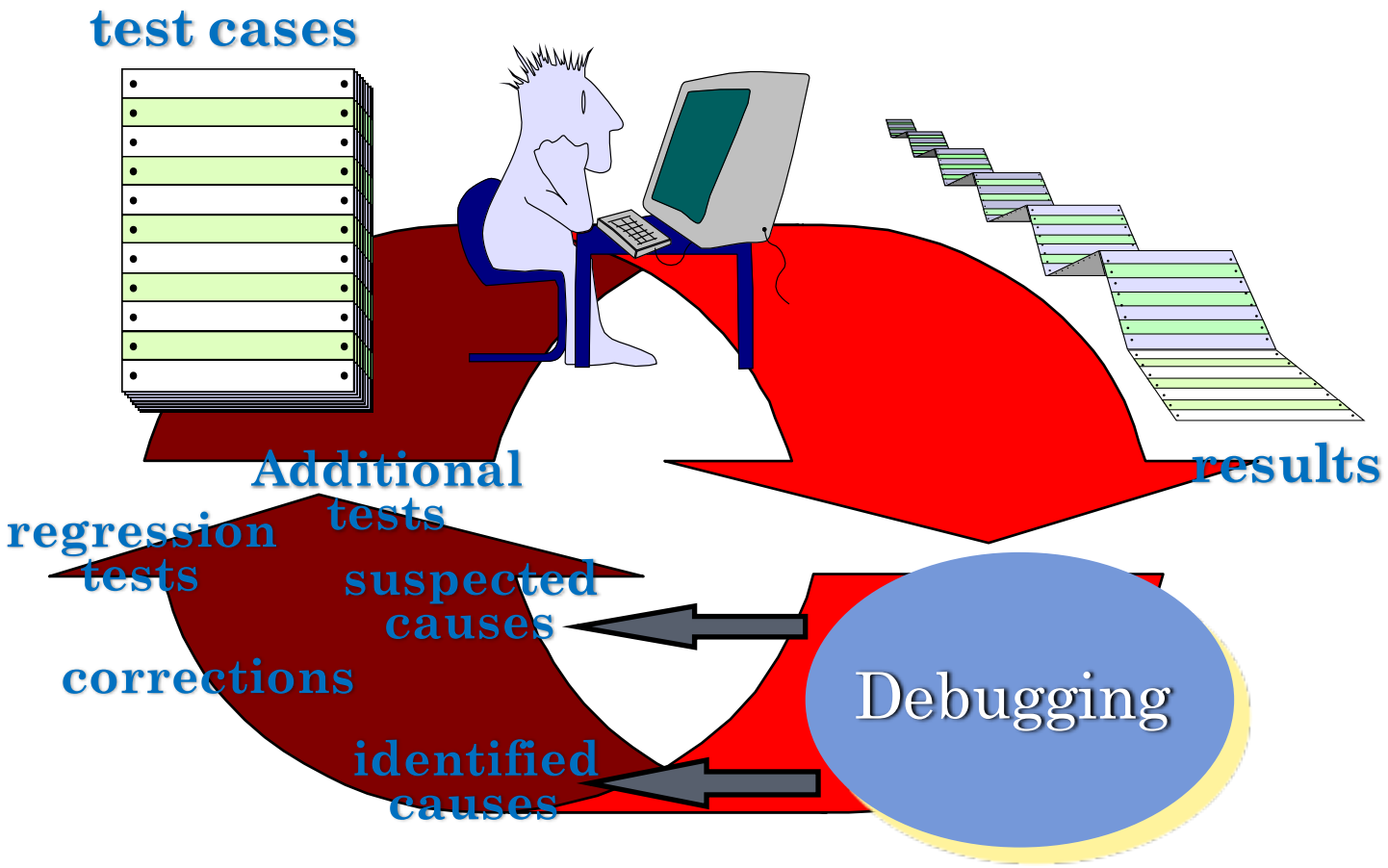
Consequences of Bugs



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

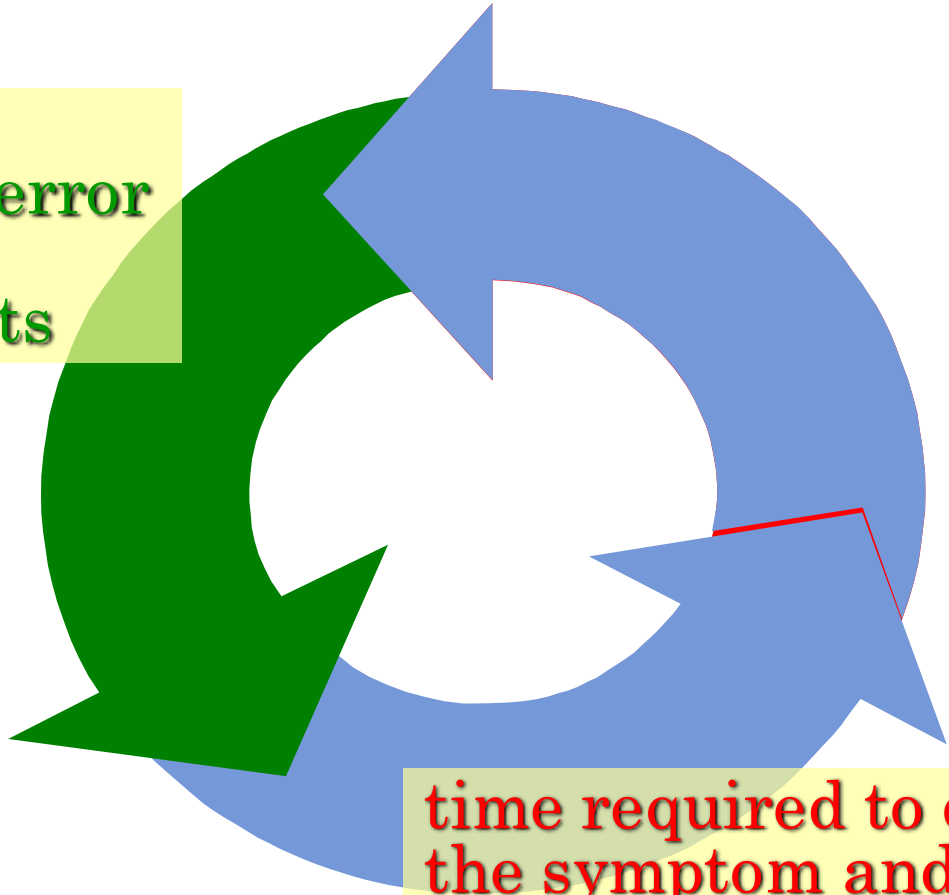
Debugging: A Diagnostic Process

The Debugging Process



DEBUGGING EFFORT

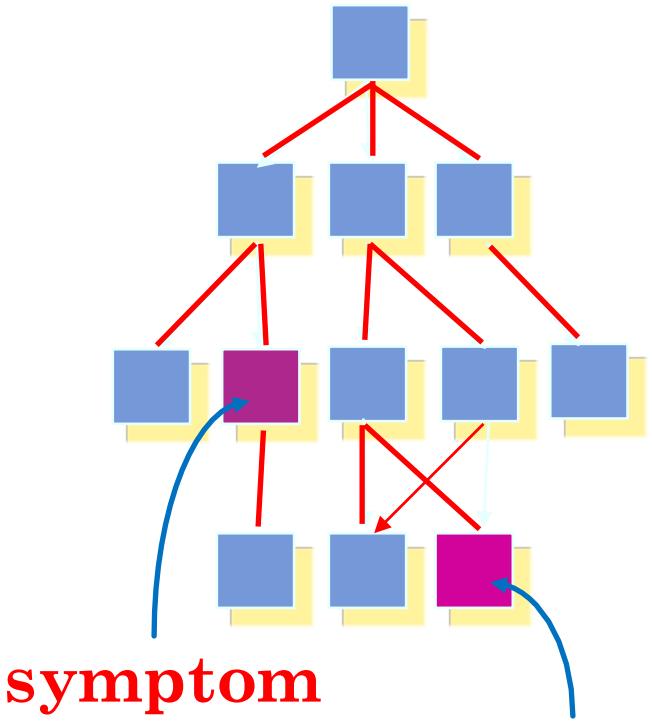
time required to correct the error and conduct regression tests



time required to diagnose the symptom and determine the cause

"The first step in fixing a broken program is getting it to fail repeatedly (on the simplest example possible)." T.Duff

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

Debugging Tactics

- The basis of debugging is to locate the problem's source by binary partitioning, through working hypotheses that predict new values to be examined
- A non-software example
  - Symptom: a lamp in my house doe not work
  - Cause hypotheses: problem lies in the lamp or local circuit breaker? No, because nothing in the house works
    - Then the problem lies in main circuit breaker or outside
    - Cause hypotheses: problems lie in outside breaker? No, because the neighborhood is not blacked out
    - ……

Debugging Techniques

- Brute force: the most common and least efficient, can used when all else fails
  - Philosophy: Let the computer find the error (find a clue from the morass of information produced by computer)
  - Methods: memory dumps, runtime traces, execution with output statements
- Backtracking: fairly common for small programs
  - Methods: manually trace backward the source code from an uncovered symptom until the site of cause
- Cause elimination by induction or deduction
  - Introduce the concept of *binary partitioning*
  - Devise "cause hypothesis" then prove or disprove the hypothesis

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.

DEBUGGING: FINAL THOUGHTS

1. Don't run off half-cocked. Think about the symptom you're seeing.
2. Use tools (e.g., dynamic debugger) to gain more insight.
3. If at an impasse, get help from others.
4. Be absolutely sure toconduct regression tests when you do "fix" the bug.

End of Chapter 14