Introduction to Software Testing

CS 3250
Software Testing

[Ammann and Offutt, "Introduction to Software Testing," Ch. 1, Ch. 2.1]

How Do You Test?

- Imagine we are testing a (typical) "divide" functionality of a calculator program.
- The divide() function takes 2 numbers that the user enters and returns the value of the 1st number divided by the 2nd number.
- The program then displays the result of the division and lets the user decide when the program should terminate.
- How would you test the divide() function? What are test case inputs? How many test cases do we need? How do you know?

```
def divide():
    n1 = int(input('Enter number1: '))
    n2 = int(input('Enter number2: '))
    return n1 / n2

end = 'n'
while end != 'y':
    print('result =', divide())
    end = input('Enter \'y\' to end the program: ')
```

How Do You Debug?

- Something is not right about the divide() function below
- The divide() function takes 2 numbers that the user enters and returns the value of the 1st number divided by the 2nd number.
- The program displays the result of the division.
- However, when the user enters
  5,2 -- the program does not display the result as 2.5
- How would you debug the divide() function? What/how to fix?

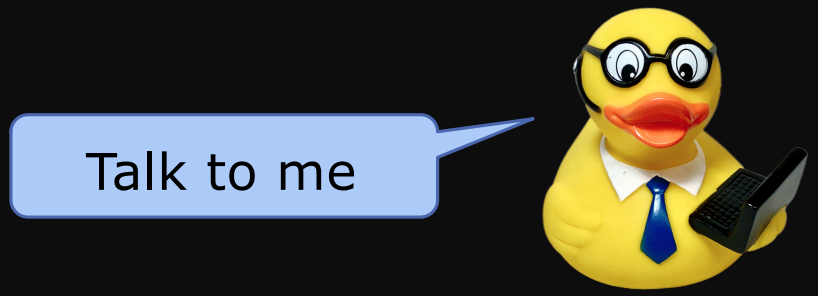
```
def divide():
    n1 = int(input('Enter number1: '))
    n2 = int(input('Enter number2: '))
    return n1 // n2

end = 'n'
while end != 'y':
    print('result =', divide())
    end = input('Enter \'y\' to end the program: ')
```

Let's Debug a (bit) Bigger Program

Refer to "Rubber Duck Debugging" Activity on the schedule page

There is a problem with `patternIndex.java`. It fails to produce the expected output.

Rubber duck debugging



(http://www.cs.virginia.edu/~up3f/cs3250/inclass/activity-debugging.html)

Think about your experience.

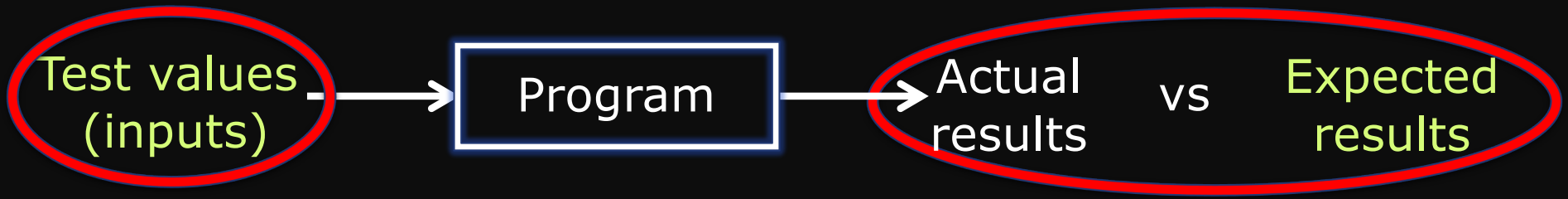
What exactly is testing?

Is testing the same as debugging?
Is debugging a testing activity?

What is Software Testing?

- Testing = process of finding input values to check against a software *(focus of this course)*

  Test case consists of test values and expected results
  Goal – reveal faults



1. Testing is fundamentally about choosing finite sets of values from the input domain of the software being tested

2. Given the test inputs, compare the actual results with the expected results

- Debugging = process of finding a fault given a failure

Testing Categories

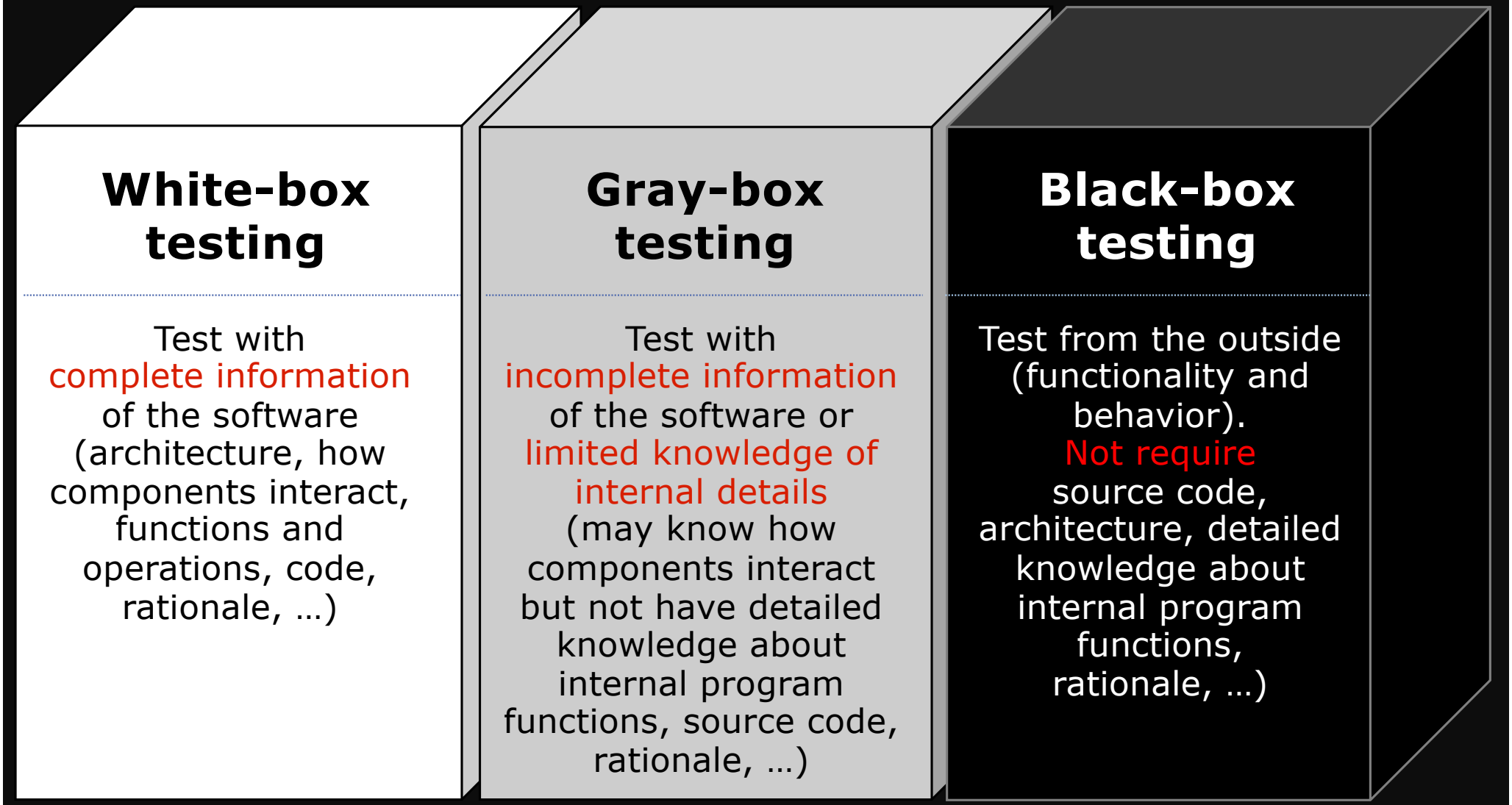
Static testing

- Testing without executing the program
  - Software inspection and some forms of analysis
  - Effective at finding certain kinds of problems such as problems that can lead to faults when the program is modified
- Inspection, walkthrough, code review, informal review, …

Dynamic testing

- Testing by executing the program with real inputs
- Unit testing, integration testing, system testing, acceptance testing, …

Testing Categories (2)



Testing Categories (3)

Functional testing

- Unit testing
- Integration testing
- System testing
- Smoke testing
- Acceptance testing (Alpha / Beta
- Agile testing
- Regression testing
- Continuous integration testing

… and many more …

Non-functional testing

- Performance testing
- Load testing
- Stress testing
- Security testing
- Compatibility testing
- Reliability testing
- Usability testing
- Compliance testing
- Conformance testing

… and many more …

Validation and Verification (IEEE)

Validation

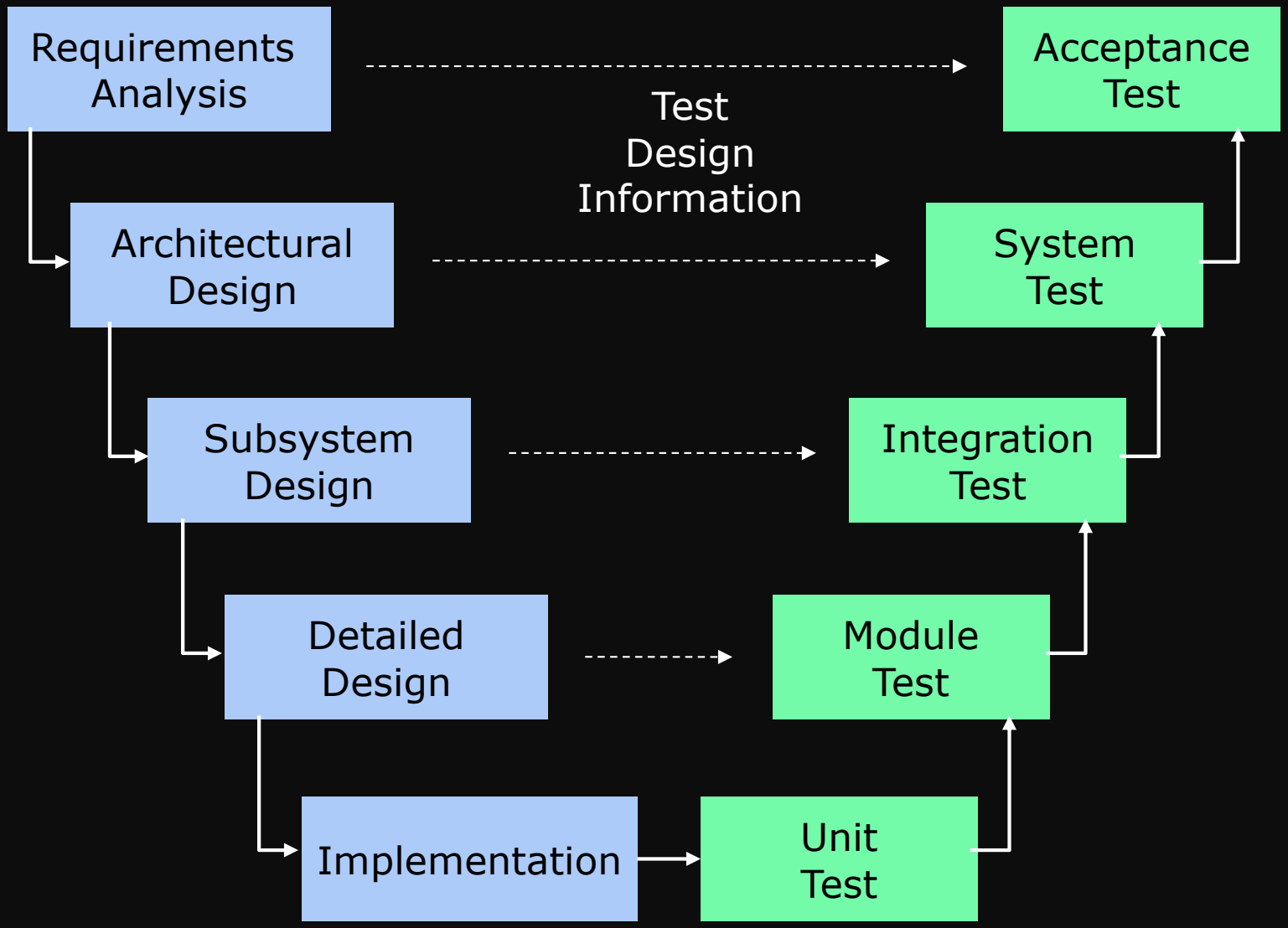
- Evaluate software at the end of software development
- Ensure compliance with the intended usage
- Done by experts in the intended usage of the software, not developers

Verification

- Evaluate software at a given phase of the development process
- Fulfill the requirements established during the previous phase
- Requires technical background on the software
- Done by developers at the various stages of development

IV&V stands for "*independent verification and validation*"

Testing and SW Development Lifecycle



Check if software does what the user needs

Check overall behavior w.r.t. specs

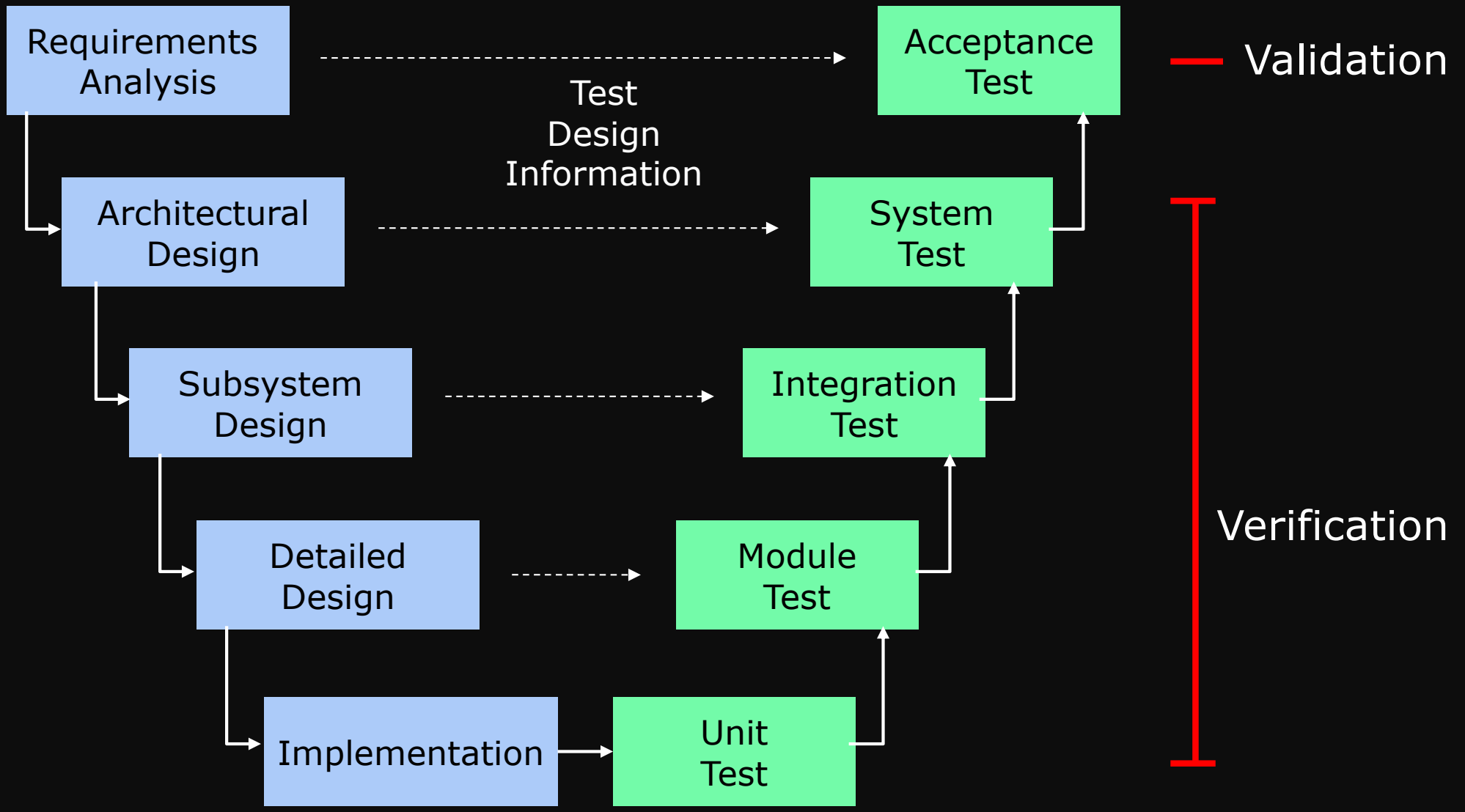
Check interface between modules in the same subsystem

Check interactions of units and associated data structures

Check each unit (method) individually

[based in part on AO, p.23]

Testing and SW Development Lifecycle



[based in part on AO, p.23]

Goals based on Test Process Maturity

Beizer's scale for test process maturity

- Level 0: There is no difference between testing and debugging
- Level 1: The purpose of testing is to show correctness
- Level 2: The purpose of testing is to show that the software does not work
- Level 3: The purpose of testing is not to prove anything specific, but to reduce the risk of using the software
- Level 4: Testing is a mental discipline that helps all IT professionals develop higher quality software

[AO, p.9]

Level 0 – Testing is Debugging

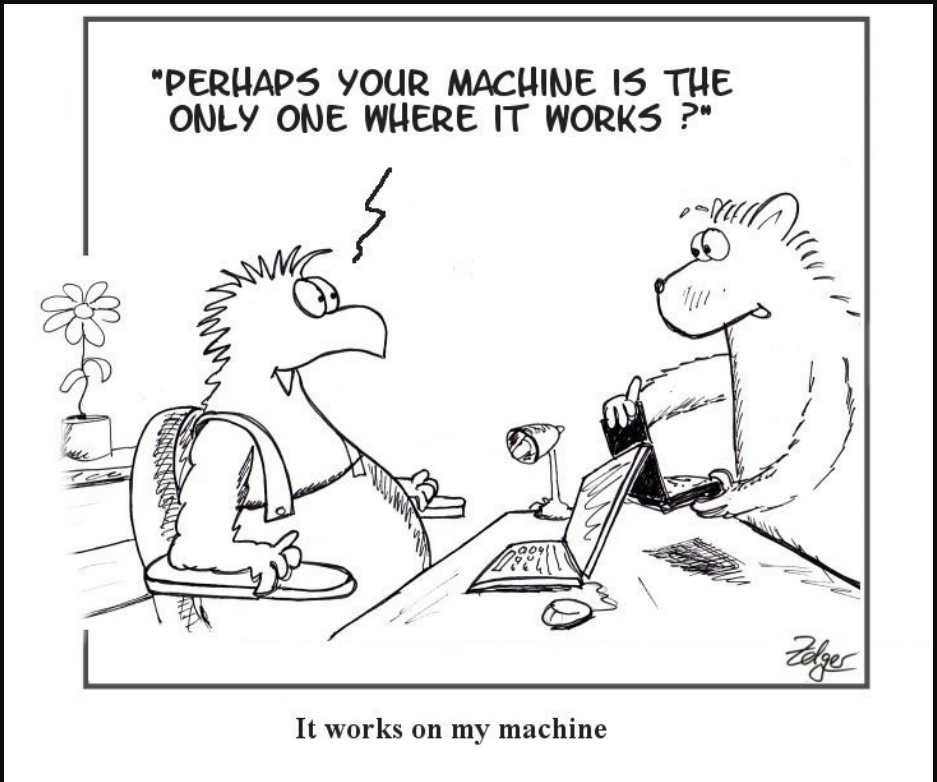
- Level 0: Testing is the same as debugging

- Not distinguish between incorrect behavior and mistakes in the program
- Not help develop software that is reliable
- Adopted by most CS students ☹
  - Get programs to compile
  - Debug with few arbitrarily chosen inputs or those provided by instructors

[image: http://softwaretestingandqa.blogspot.com/2007/12/software-bug.html]

Levels 1 – Software Works

- Level 1: To show correctness (developer-biased view)

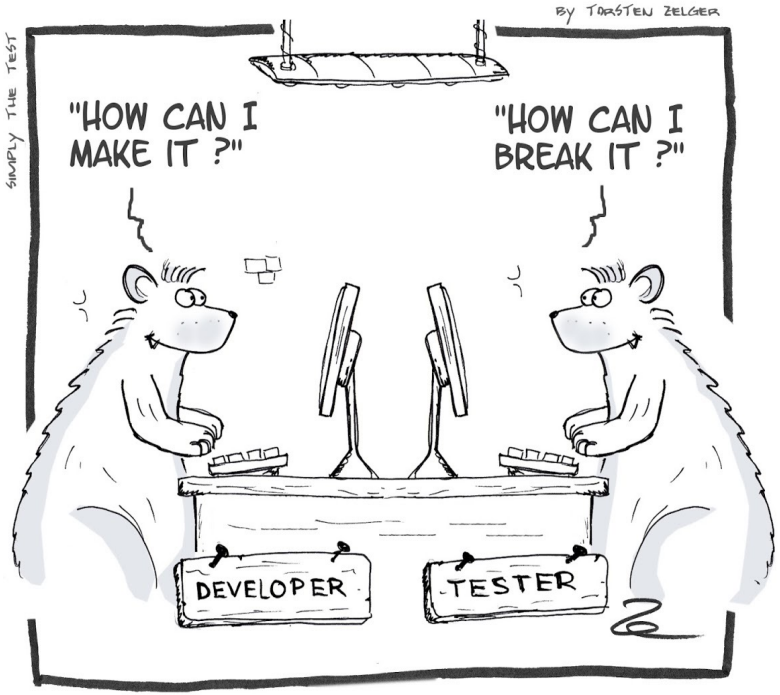


- Correctness is impossible to establish or demonstrate
- What do we know if "no failures"?
  - Good software?
  - Bad tests?
- No strict goal, no stopping rule or formal test technique
- No quantitatively way to evaluate; test managers are powerless

[image: http://simply-the-test.blogspot.com/2010/]

Levels 2 – Software Doesn't Work

- Level 2: To show failure (tester-biased view)
  - A negative view puts testers and developers into an adversarial relationship – bad team morale
  - What do we know if "no failures"?



They were not so much different, but they had different goals

[image: http://simply-the-test.blogspot.com/2010/]

"Mature" Testing

Correctness cannot generally be achieved or demonstrated through testing.

Testing can only show the presence of failure, not the absence.

Developers and testers should be on the same boat.

How can we move to a team approach?

Levels 3 – Risk Reduction

- Level 3: To reduce the risk of using the software
  - There are risks when using software
    - Some may be small with unimportant consequences
    - Some may be big with important consequences, or even catastrophic
  - Testers and developers cooperate to reduce risk

Levels 4 – Quality Improvement

- Level 4: To increase quality of the software
  - Testing should be an integral part of the development process
  - Testers become technical leaders → measuring and improving software quality
    - Help developers improve the ability to produce quality software
    - Train developers
  - Testers and developers cooperate to improve the quality
  - Example: Spell checker
    - Purpose: to improve our ability to spell
    - Change of mindset:

      "find misspelled words" → "improve ability to spell"

How "Mature" is Your Testing?

Are you at level 0, 1, 2, 3, or 4?

We hope to train you to become "change agents" (level 4)

Principles of Software Testing

Why testing is so hard

1. Exhaustive testing is impossible
2. We need to know when to stop testing
3. There is no silver bullet in software testing
4. Faults happen in some places more than others
5. Bug-free software does not exist
6. Testing is context-dependent
7. Verification is not validation

Tactical Goals: Each Test

"If you don't know why you're conducting each test, it won't be very helpful" – Jeff Offutt

- What is objective and requirement of each test?
- What fact does each test try to verify?
- What are the threshold reliability requirements?
- What are the planned coverage levels?
- How many tests are needed?

Wrap-up

- A tester's goal is to eliminate faults as early as possible
- Testing
  - Improves software quality
  - Reduce cost
  - Preserve customer satisfaction
- **What's next?**
  - Fault, error, failure
  - Reachability, Infection, Propagation, and Revealability (RIPR) model