



Carnegie
Mellon
University

 Electrical & Computer
ENGINEERING

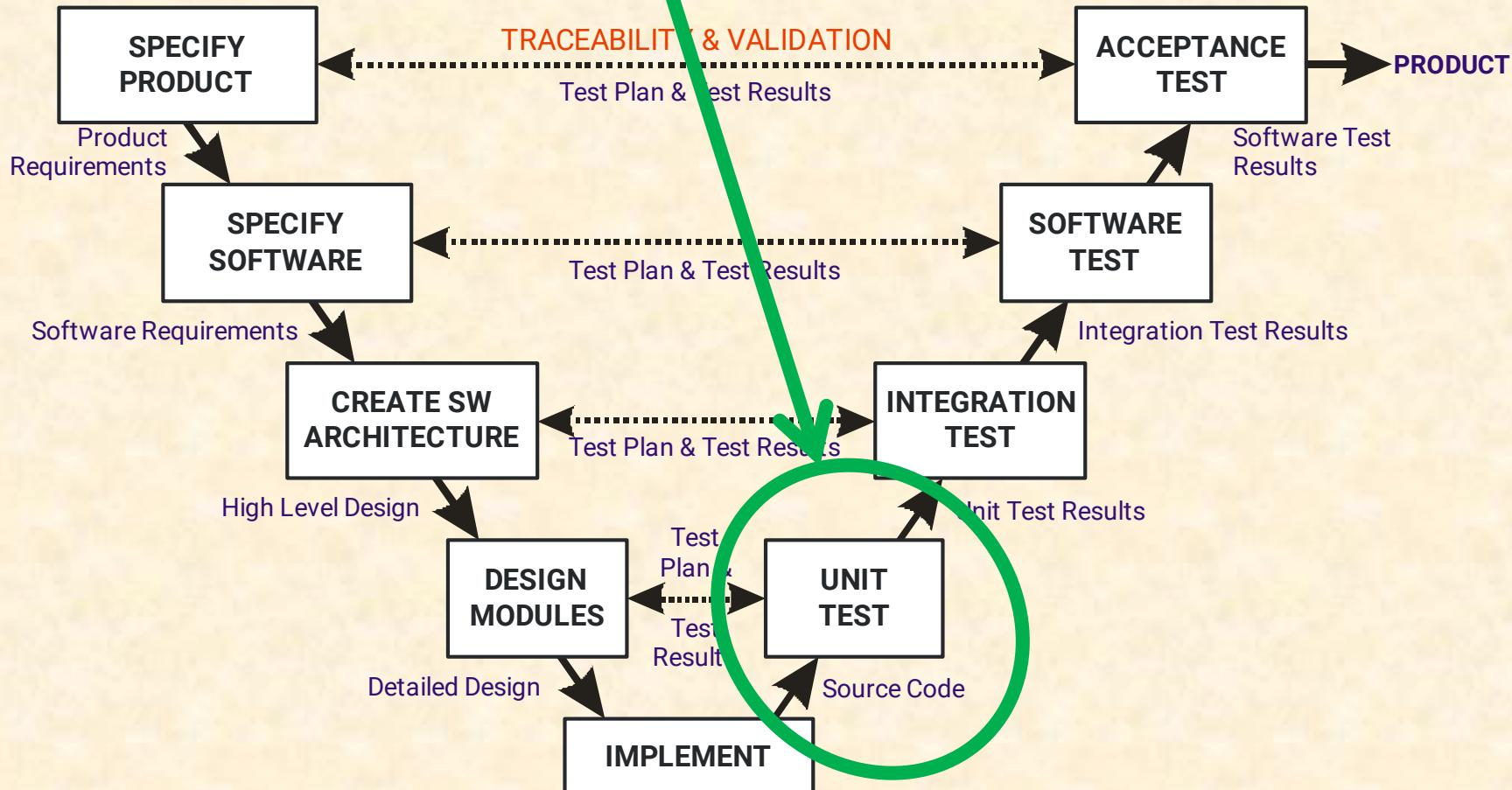
Unit Testing

Prof. Philip Koopman

“Quality is free, but only to those who are
willing to pay heavily for it.”

— DeMarco & Lister

YOU ARE HERE



Unit Testing

■ Anti-Patterns:

- Only system testing
- Testing only “happy paths”
- Forgetting to test “missing” code

■ Unit testing

- Test a single subroutine/procedure/method
 - Use low level interface (“unit” = “code module”)
- Test both based on structure and on functionality
 - White box structural testing + Black box functional testing
- This is the best way to catch boundary-based bugs
 - Much easier to find them here than in system testing

Test cases:

a = 0; b = 0;

a = -1; b = +1;

...

```
uint16_t proc(uint16_t a, uint16_t b)
{
    ....
    return(result);
}
```

Expected Test Results:

a = 0; b = 0; ==> 0

a = -1; b = +2; ==> 1

...

Black Box Testing

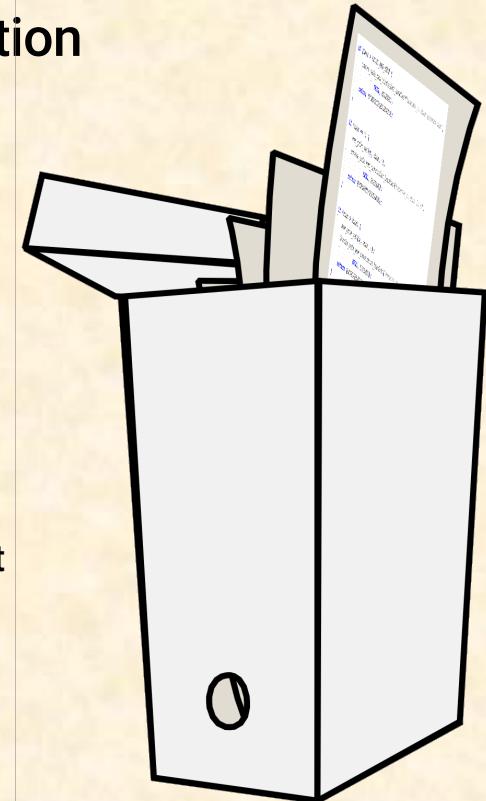
- Tests designed based on behavior
 - But without knowledge of implementation
 - “Functional” or behavioral testing
- Test the what, but not the how
 - Example: cruise control black box test
 - Test operation at various speeds
 - BUT, no way to tell if special cases in code have been tested
 - Advantage: can be written only based on requirements or design
 - Disadvantage: difficult to exercise all code paths
- Black box Unit Testing
 - Tests based on detailed design (statechart, flowchart)



<https://goo.gl/wJeZ56>

White Box Testing

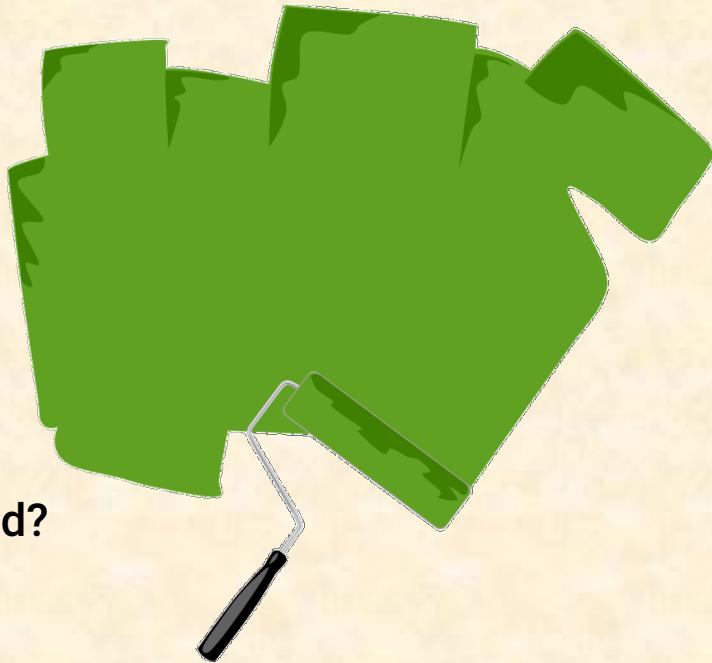
- Tests designed with knowledge of software implementation
 - Often called “structural” testing
 - Sometimes: “glass box” or “clear box”
- Idea is to exercise software knowing how it is written
 - Example: cruise control white box test
 - Exercise every line of code
 - » Tests that exercise both paths of every conditional branch statement
 - Test operation at every point in control loop lookup table
 - Advantage: helps getting high structural code coverage
 - Disadvantage: doesn’t prompt coverage of “missing” code
 - E.g., missing special case, missing exception handler



Unit Testing Coverage

Coverage is a metric for how thorough testing is

- Function coverage
 - What fraction of functions have been tested?
- Statement coverage
 - What fraction of code statements have been tested?
 - (Have you executed each line of code at least once?)
- Branch coverage (also Path Coverage)
 - Have both true and false branch paths been exercised?
 - Includes, e.g., testing the false path for `if (x) { ... }`
- MCDC coverage (next slide)
- Getting to 100% coverage can be tricky
 - Error handlers for errors that aren't supposed to happen
 - Dead (unused) code that should be removed from source



MCDC Coverage

■ Modified Condition/Decision Coverage (MC/DC)

- Used by DO-178 for critical aviation software testing
- Exercise all ways to reach all the code
 - Each entry and exit point is invoked
 - Each decision tries every possible outcome
 - Each condition in a decision generates all outcomes
 - Each condition in a decision is shown to independently affect the outcome of the decision
- For example: “if ($A == 3 \parallel B == 4$)” → you need to test at least
 - $A == 3 ; B != 4$ (A causes branch, not masked by B)
 - $A != 3 ; B == 4$ (B causes branch, not masked by A)
 - $A != 3 ; B != 4$ (Fall-through case)
 - $A == 3 ; B == 4$ is NOT tested because it’s redundant (no new information gained)
- Might need trial & error test creation to generate 100% MCDC coverage

MC/DC : EXAMPLE
 $a \& b \& c$

Test case	a	b	c	outcome
1	True	True	True	True
2	True	True	False	False
3	True	False	True	False
4	True	False	False	False
5	False	True	True	False
6	False	True	False	False
7	False	False	True	False
8	False	False	False	False
9	True	True	True	True
10	False	True	True	False

<https://www.youtube.com/watch?v=DivaWCNohdw>

Unit Testing Coverage Strategies

■ Boundary tests:

- At borders of behavioral changes
- At borders of min & max values, counter rollover
- Time crossings: hours, days, years, ...

■ Exceptional values:

- NULL, NaN, Inf, null string, ...
- Undefined inputs, invalid inputs
- Unusual events: leap year, DST change, ...

■ Justify your level of coverage

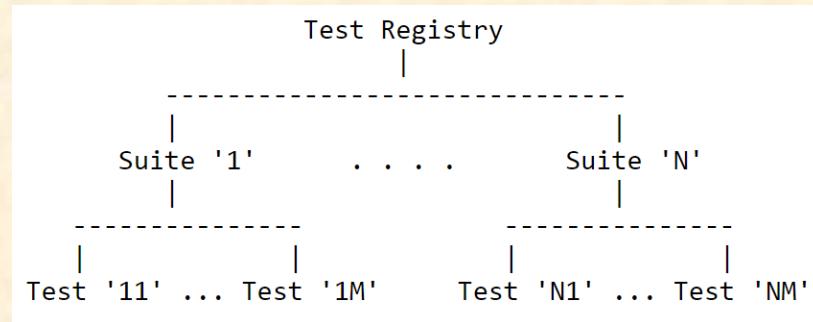
- Trace to unit design
- Get high code coverage
- Define strategy for boundary & exception coverage



Unit Testing Frameworks

■ Cunit as an example framework

- Test Suite: set of related test cases
- Test Case: A procedure that runs one or more executions of a module for purpose of testing
- Assertion: A statement that determines if a test has passed or failed



<http://cunit.sourceforge.net/doc/introduction.html>

■ Test case example: (http://cunit.sourceforge.net/doc/writing_tests.html#tests)

```
int maxi( int i1, int i2)
{ return (i1 > i2) ? i1 : i2; }

...
void test_maxi (void)
{ CU_ASSERT(maxi(0,2) == 2); // this is both a test case + assertion
  CU_ASSERT(maxi(0, -2) == 0);
  CU_ASSERT(maxi(2,2) == 2); }
```

Best Practices For Unit Testing

■ Unit Test every module

- Use high coverage combination of white box & black box
- Use a unit testing framework
 - Multiple simple tests better than one huge, complex test
- Get good coverage of data values
 - Especially, validate all lookup table entries

■ Unit Testing Pitfalls

- Creating test cases is a development effort
 - Code quality for test cases matters; test cases can have bugs!
- Difficult to test code can lead to dysfunctional “unit test” strategies
 - Breakpoint debugging is not an effective unit test strategy
 - Using Cunit to test 100K lines of code is not really unit testing
- Pure white box testing is “doomed to succeed” (neglects “missing” code)
- Don’t substitute unit tests for peer reviews and static analysis



<https://goo.gl/SjzaBm>

Your application is a special snowflake



Expert

Excuses for Not Writing Unit Tests

O RLY?

@ThePracticalDev

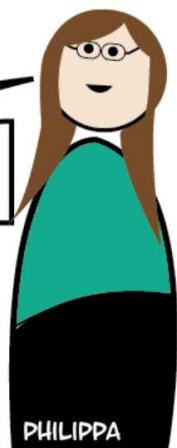


WHAT'S MODIFIED CONDITION /
DECISION COVERAGE TESTING?

HERE'S AN
ANALOGY

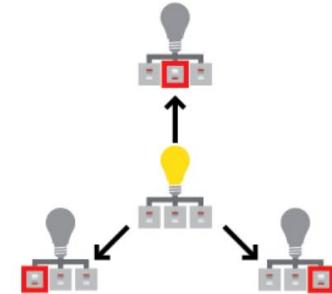
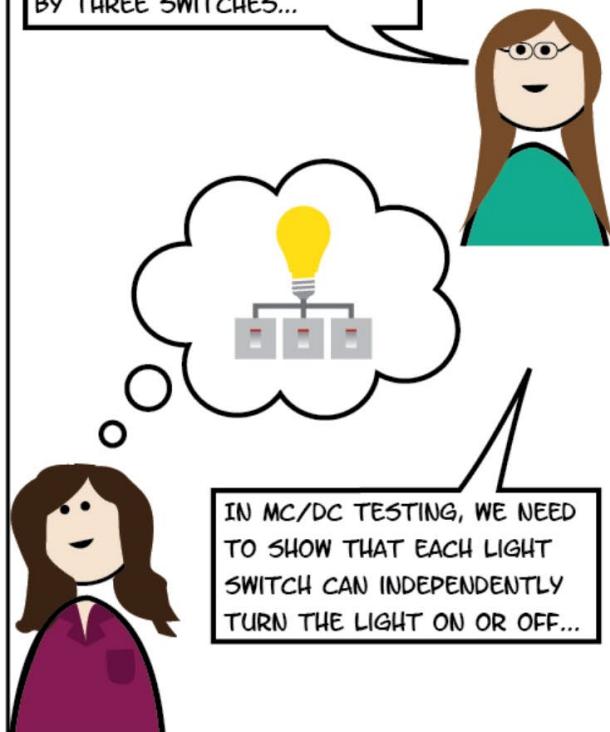


LUCY



PHILIPPA

IMAGINE A LIGHT CONTROLLED
BY THREE SWITCHES...



HOW DOES THAT APPLY
TO SOFTWARE?



THE LIGHT CORRESPONDS TO
THE DECISION AND THE
SWITCHES CORRESPOND TO
CONDITIONS

