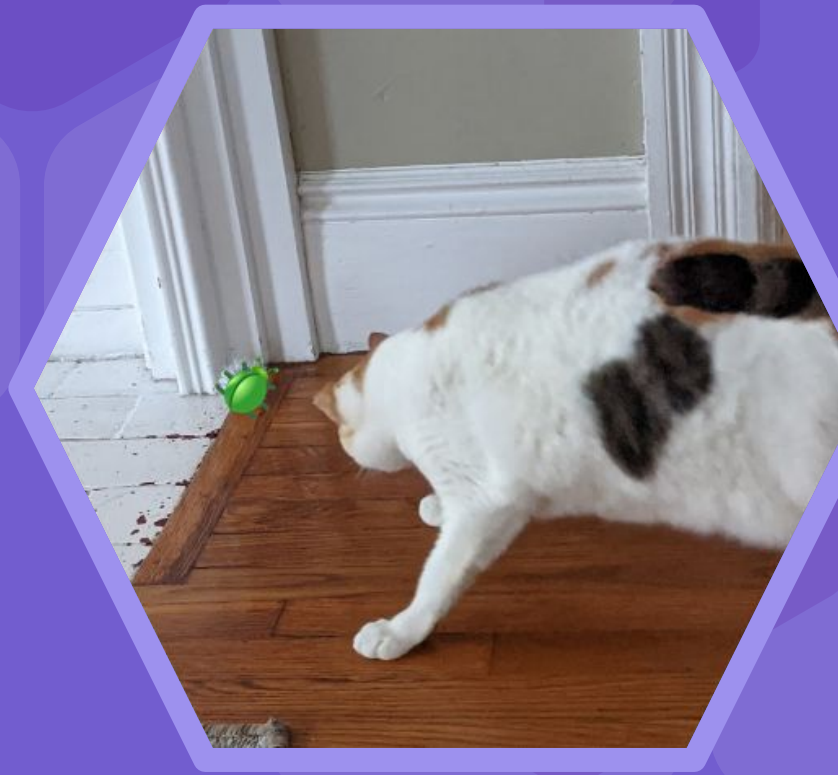
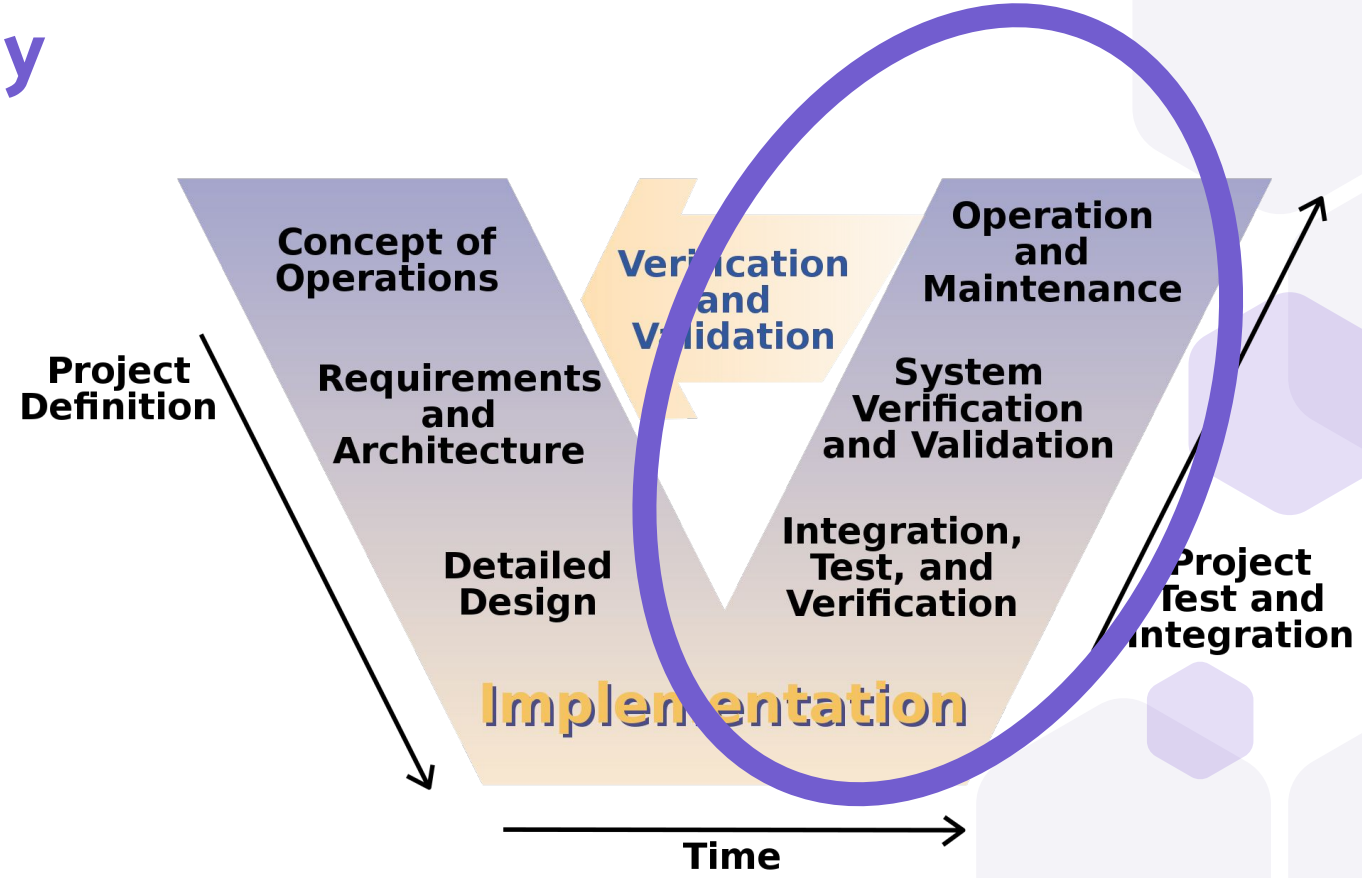
Testing and debugging



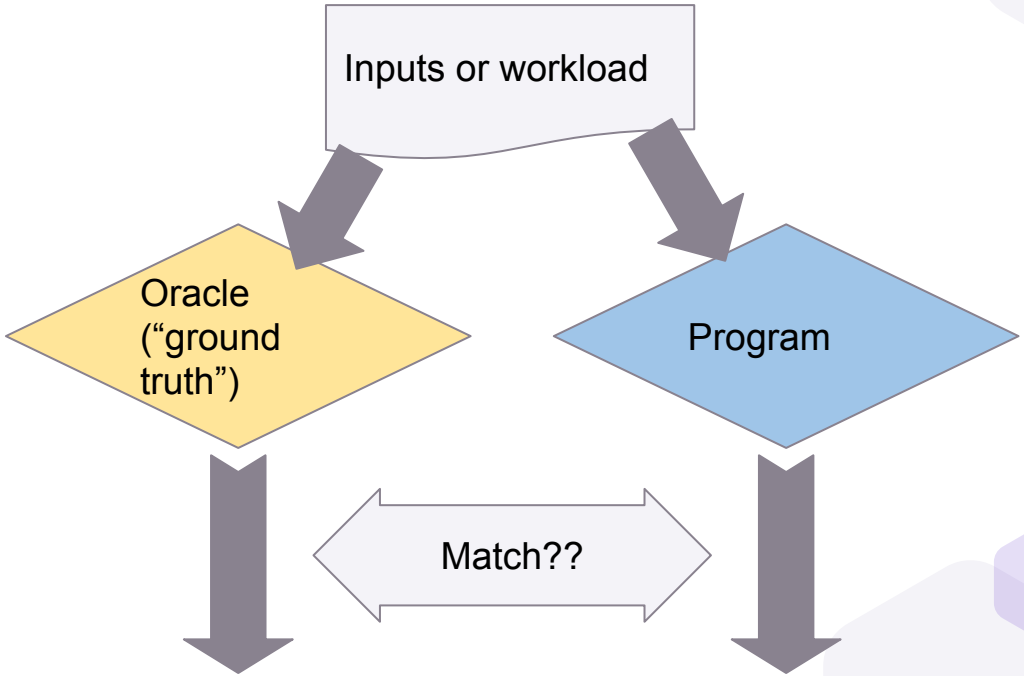
Schedule your capstone meetings!

There is a post on Ed

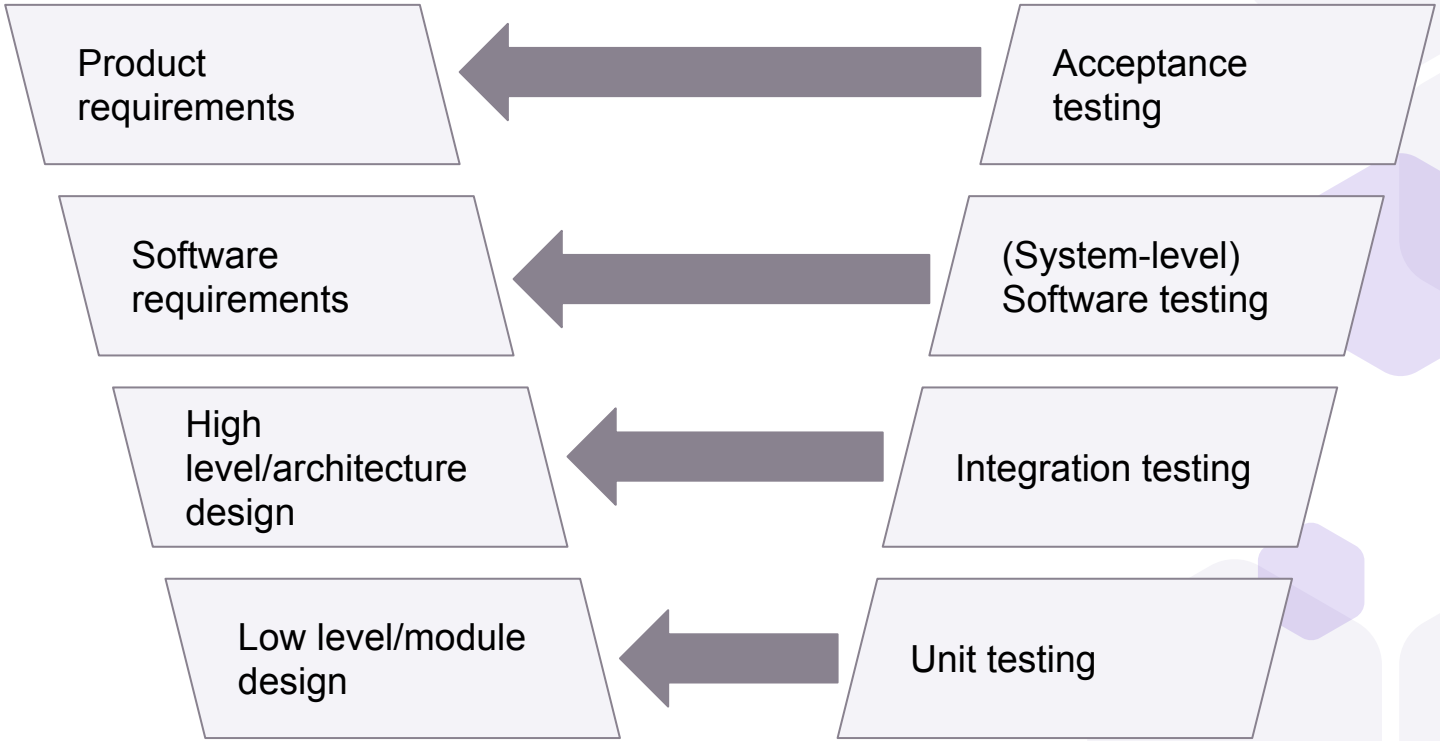
Today


Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Project Definition
Time

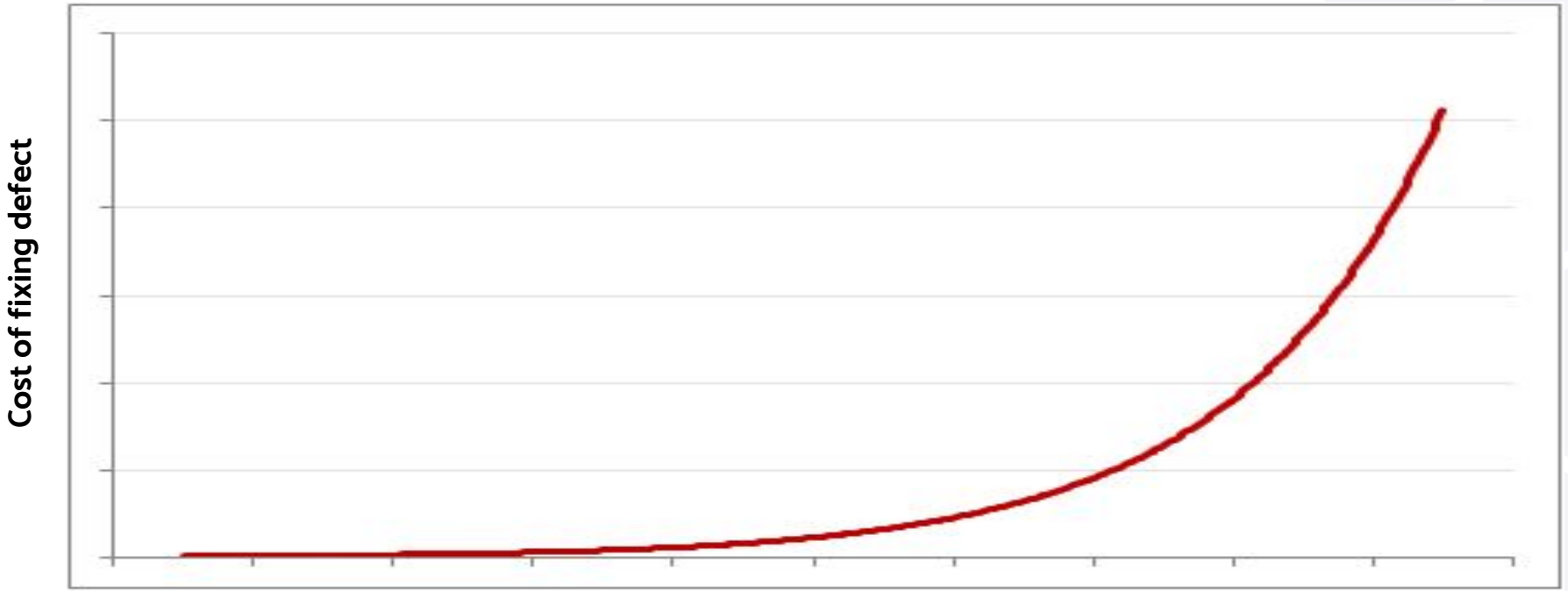
What is testing



Types of testing in the V model



Why not just system level testing?



Unit testing

Check correctness of a module

One unit test = test a single function/method/path

- Cannot test even single function calls exhaustively - consider f(int x, int y, int z)
- Best place to test edge case values

Both structural and functional testing

Functional vs. structural testing

Functional

"Black box" testing

No underlying knowledge of code

Example goal: exercise every requirement for module

Pros:

The requirements are what really matters

Can't be biased by knowing the structure of code

Expensive to try to white box test

Structural

"White box" testing

Knowledge of structure of code - guides testing

Example: exercise every line of code in function call

Pros:

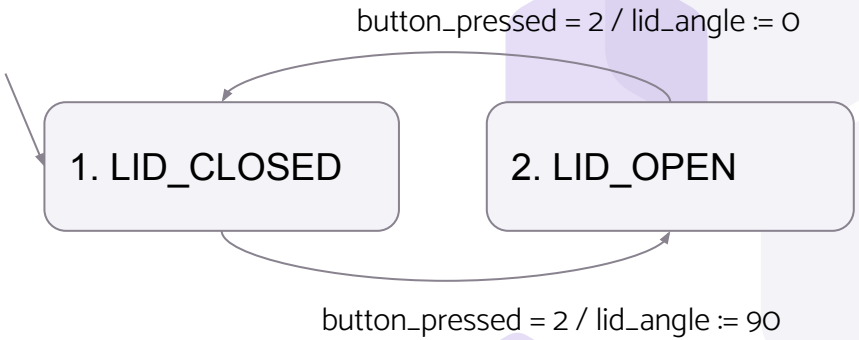
Possibly a higher chance of catching bugs at edge cases

More guidance on what values to use

What are the tradeoffs between black box and white box testing?

How to unit test an implementation based on FSM?

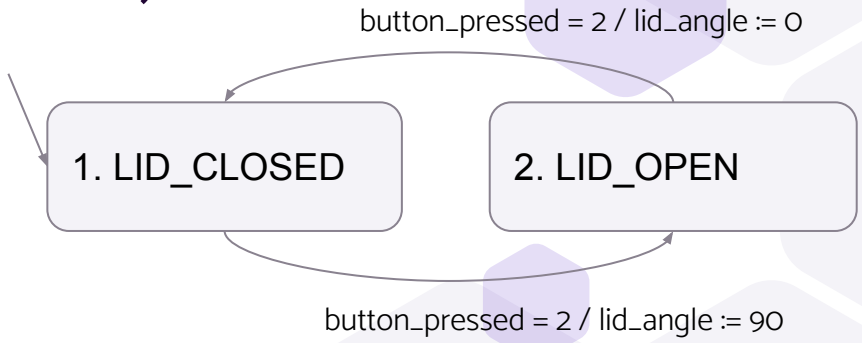
```
state update_fsm(state current_state, int button_pressed, ...) {
  state next_state;
  switch(current_state) {
    case LID_CLOSED:
    if (button_pressed == 2) {  // T 1-2
      set_lid_angle(90);
      next_state = LID_CLOSED;
    } else {
      next_state = current_state;
    }
    break;
    case LID_OPEN:
    ...
    break;
    default:
      error("wrong state!");
  }
  return next_state;
}
```



Want to test update_fsm's implementation as-is (without making changes to it)

Test for transition 1-2

```
end_state = update_fsm(LID_CLOSED,
2)
assert(end_state == LID_OPEN)
lid_angle == 90
```



Mock out functions

```
// #define TESTING // uncomment to test
#ifndef TESTING // means TESTING is not defined
void set_lid_angle(int ang) { ...normal operation … }
#else
int test_lid_angle;
void set_lid_angle(int ang) { test_lid_angle = ang; }
#endif
```

Updated test of FSM transition 1-2

```
end_state = update_fsm(LID_CLOSED,
2)
assert(end_state == LID_OPEN);
assert(test_lid_angle == 90);
```

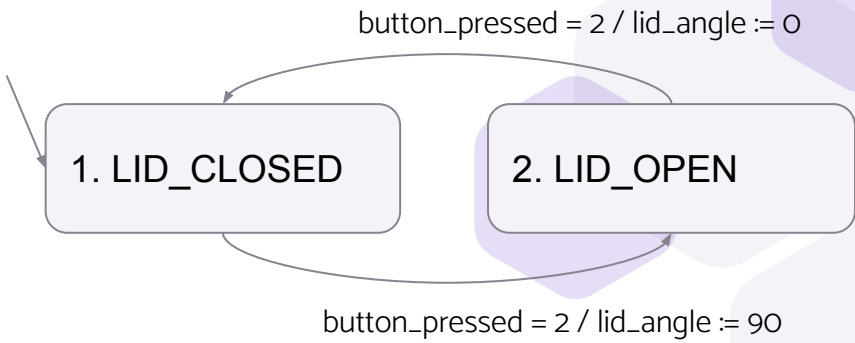
Edge case/unexpected inputs

What should this do?

```
update_fsm(LID_CLOSED, 4)
```

What about this?

```
update_fsm(LID_CLOPEN, 1)
```



Break

Coverage

Notion of how completely a piece of code has been tested with a particular set of tests, with respect to a specific metric

Examples:

- What % of requirements have been tested?
- What % of lines of code have been tested?

100% coverage does **not** mean 100% tested, but it's a start to assess testing thoroughness

White box testing guided by coverage

Branch (aka decision) - *for every branch (e.g. if-statement), is there at least one test case that evaluates that branch to true and one that evaluates it to false?*

Condition - *like branch coverage, but looking at conditions* within *branches (e.g. looking at x > 0 and y == 2 separately rather than just x > 0 || y == 2)*

Path - *is there a test case that exercises every unique* path *through the code (as opposed to considering each branch independently)*

Branch coverage

```
if (x == 3 && y < 0 ) {
 // do something;
} else {
  // do something else
}

q = x + z;

if (q < y) {
  if (x == z) {
    // do another thing
  }
}
```

	x==3 && y < 0	X + z < y	x == z
(3, -1, 3)	true	false	n/a
(3, 4, 0)	false	true	false
(2, 5, 2)	false	true	true

Condition coverage

```
if (x == 3 && y < 0 ) {
 // do something;
} else {
  // do something else
}

q = x + z;

if (q < y) {
  if (x == z) {
    // do another thing
  }
}
```

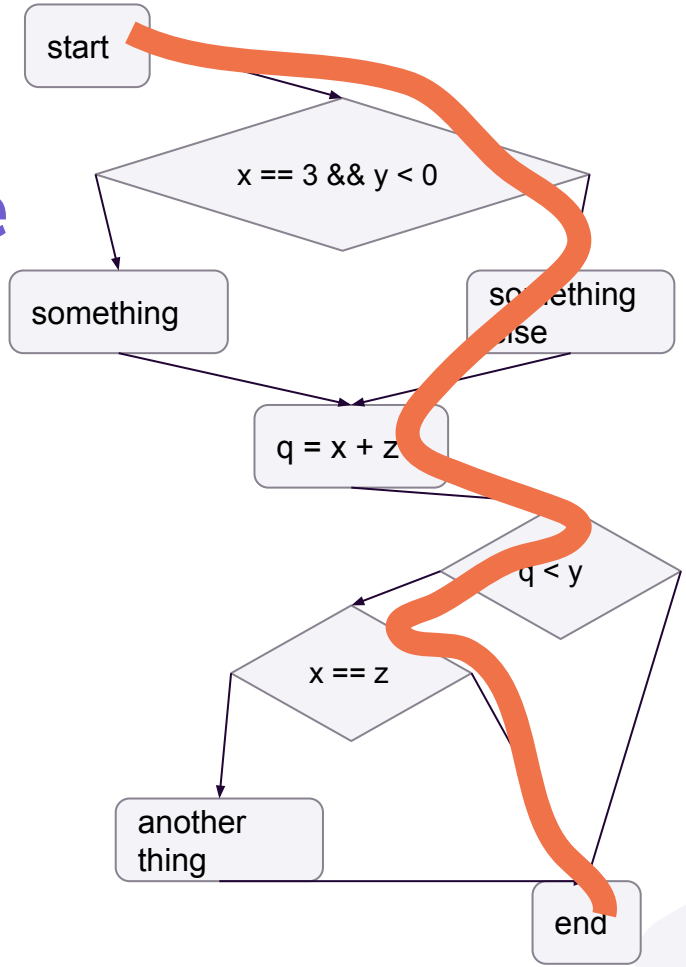
	x==3	y < 0	X + z < y	x == z
(3, -1, 0)	true	true	false	n/a
(0, 1, 0)	false	n/a	true	true
(3, 4, 0)	true	false	true	false

Path coverage

```
if (x == 3 && y < 0 ) {
 // do something;
} else {
  // do something else
}

q = x + z;

if (q < y) {
  if (x == z) {
    // do another thing
  }
}
```



Modified Condition/Decision Coverage (MC/DC)

A more comprehensive coverage metric required by some software safety standards

- Each entry and exit point is invoked
- Each decision takes every possible outcome <- branch coverage
- Each condition in a decision takes every possible outcome <- condition coverage
- **Each condition in a decision is shown to independently affect the outcome of the decision**

Each condition in a decision is shown to independently affect the outcome of the decision

Pick values to hold all but one condition constant. Does changing the other condition affect the outcome of the decision?

```
(x + y) == 3 && (y < 0 || x == 2)
```

x	y	x + y == 3	y < 0	x == 2	decision
4	-1	true	true	false	true
3	-1	false	true	false	false
1	2	true	false	false	false
2	1	true	false	true	true

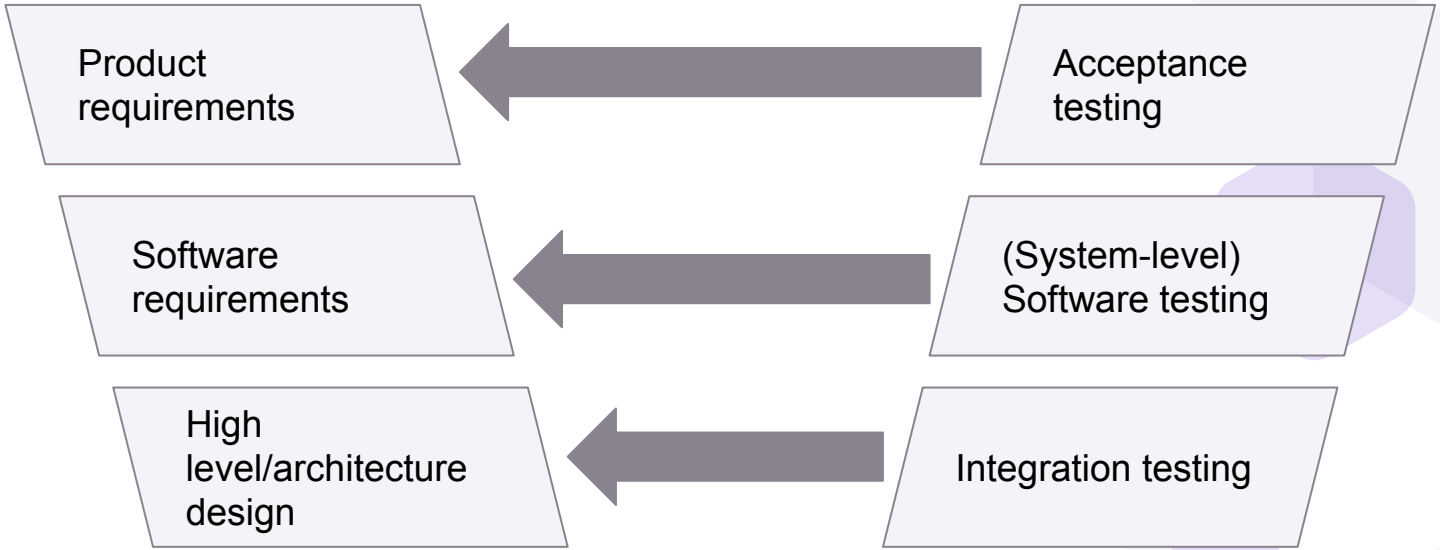
Unit testing summary

Cheaper to catch defects here than at any other stage of testing

Perform structural (white-box) or functional (black-box) testing on modules/components/functions

Assess completeness of testing with coverage

Rest of the V


Product requirements
Acceptance testing
Software requirements
(System-level) Software testing
High level/architecture design
Integration testing

Integration testing

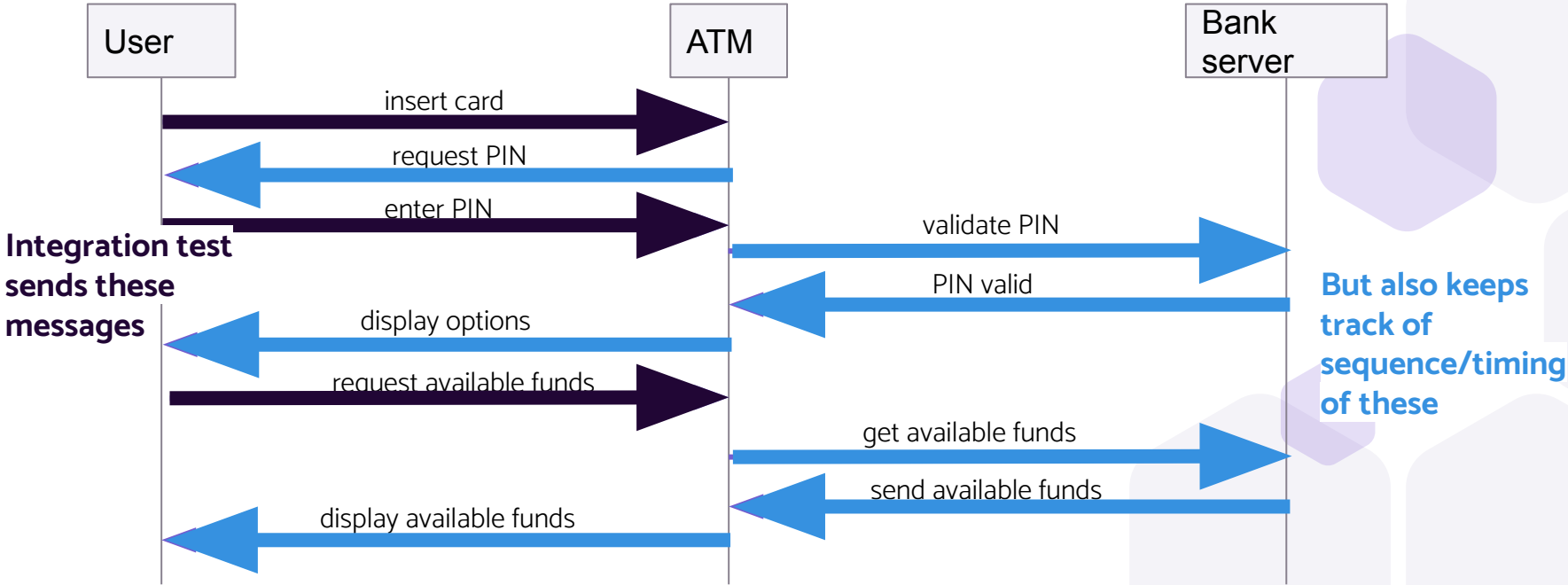
Use high level design (architecture diagram and sequence diagrams) to test interfaces between modules/components

- Test every interface (message format, correctness of values)
- Test timing and sequence of messages sent
- Test that unexpected messages are handled

Assume modules are performing individual duties correctly (**why?**) and just test the *communication* between them

Sequence diagram test example

Scenario: check available funds at ATM



Message formats

Messages are data structures with multiple fields

Example: message to validate pin may have fields

- Message header (message #, timestamp, origin ID, etc)
- Message type (“validate” encoded into bits)
- (Encrypted) user ID
- (Encrypted) PIN
- Checksum

Part of integration testing is checking that message formats are handled correctly

What are some coverage criteria for integration tests?

System testing

- Tests all system requirements
  - Tests entire system: do inputs at external interfaces cause the correct overall behavior?
- Software testing: tests from software POV
- Acceptance testing: tests from customer POV
- Expensive to instrument, very expensive to repair
  - Bug at system level is a process failure

Break

Other kinds of testing and tests

Smoke testing - *turn the system on and see if the system works at all (way to check if rest of the system is worth testing)*

Exploratory testing - *skilled tester exercises the system by hand*

Beta test - *product tested by a representative group of users*

Regression test - *did bug fix introduce new bugs?*

Robustness test - *does system hold up to invalid inputs?*

Security test - *can an attacker compromise the system?*

Performance test - *bandwidth, speed, data usage...*

Testing plans

What should be tested?

- What level(s) of testing? (unit, integration, system)
- What kind(s) of testing for each level?

How should it be tested?

- Define testing frameworks, mock functions
- Make an argument for sufficient isolation from interference

How thoroughly should you test?

- Define coverage goals

Accountability

Testing plan should be written *before* testing

Failing tests should be reported, with a plan to triage/address them

> Diagnosing and fixing failing tests is an art in itself *(but good methodology/defects caught at unit level/clean and intentional testing helps!)*

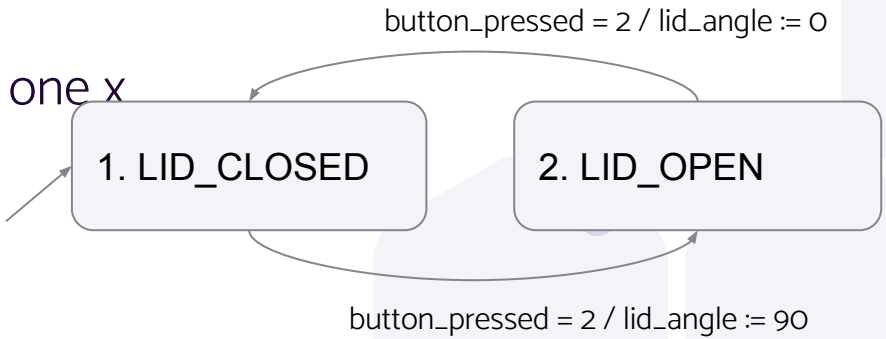
How do you know testing matches design at each level?

Traceability applies to tests too!

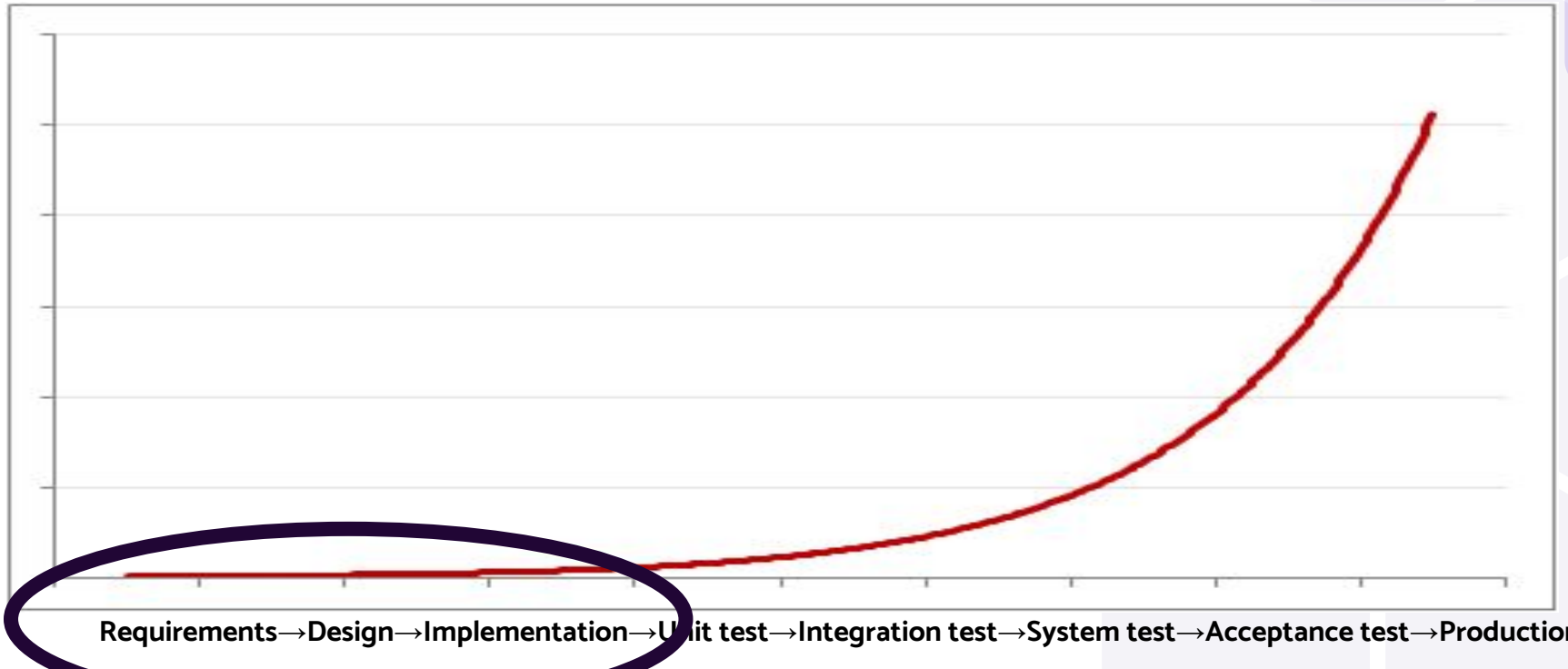
Example: trace unit test to FSM

- Number each test - put each number in row of traceability matrix
- Put each transition (and non-transition!) in columns of traceability matrix
- Make sure each column has at least one x

	1--	1-2	2--	2-1
Test 1	x			
Test 2			x	
Test 3		x		



What about this part of the chart?



Peer reviews

Structured meetings for people to review artifacts

Catch defects/discrepancies early

Can be done at every stage (requirements, design, implementation, test plan)

Healthy projects find more than half of all defects in peer review!

Fagan-style inspections

Participants have familiarity with project

Producer explains artifact but is not present for review

Roles assigned (reader, moderator, recorder)

Identify defect and move on

Give list of defects to producer and rework

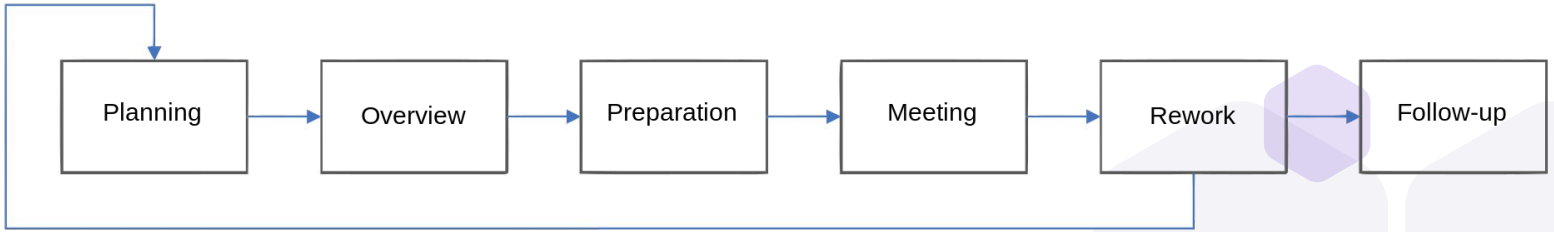


Image source

Peer review best practices

Review the artifact, not the producer

Limit meeting length (<2 hours)

Have clear roles

Have set goals (checklist); agreed upon beforehand

Do not fix problems

Inspect early and often

Bonus: what about hardware debugging?

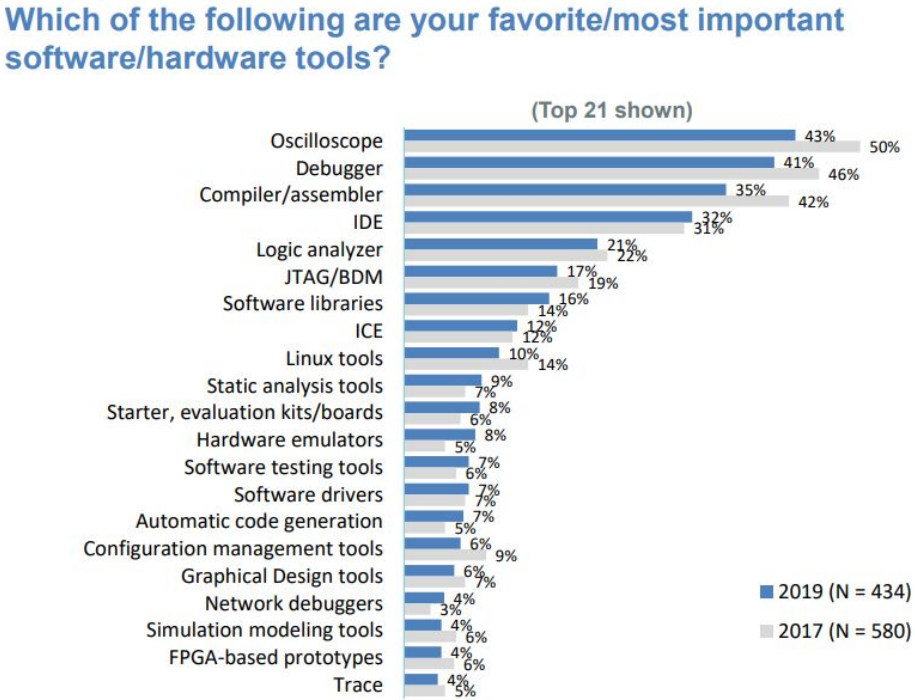


Image source

Oscilloscopes

View graphs of electrical signals

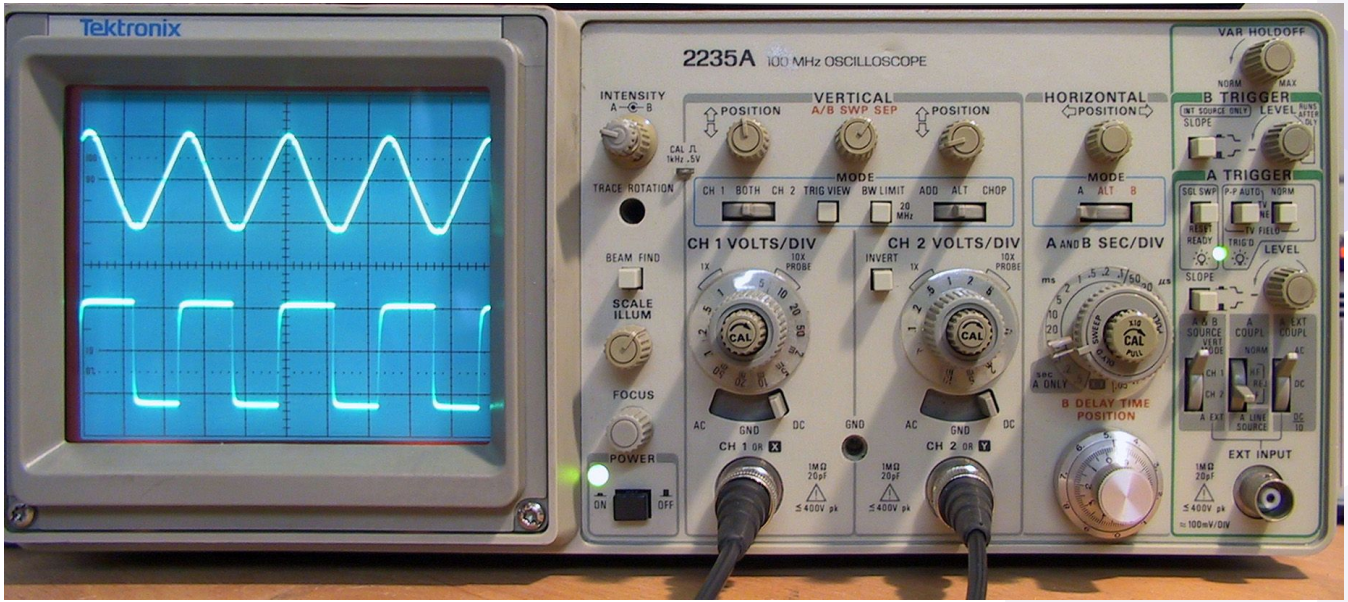


Image source

Hardware debuggers

Connect directly to pins of chip to debug at the instruction level

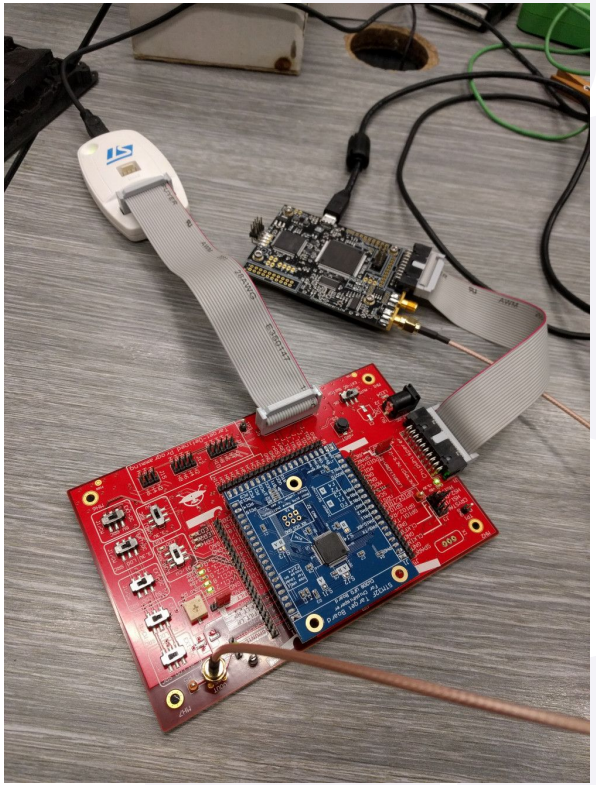


Image source

Logic analyzers

View graphs of digital signals

Can sometimes do advanced analysis and timing comparison

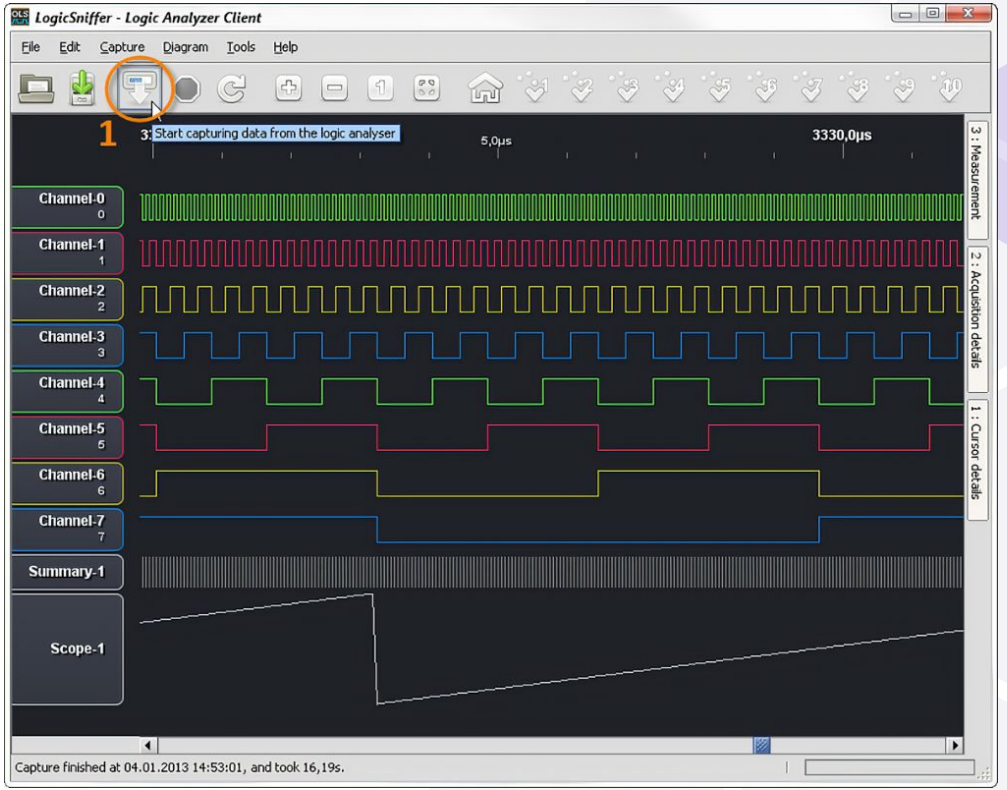


Image source

Summary

Testing should be done for every level of the V model; should trace back to left side

Coverage helps set goals for how much to test

The earlier the testing, the cheaper (and peer reviews are the cheapest of all!)