System Test

Introduction to Systems Engineering
I2ISE

Here’s a fact about test

Testing can only show the *presence* of errors, never their *absence*

- What does this mean? What are the consequences?

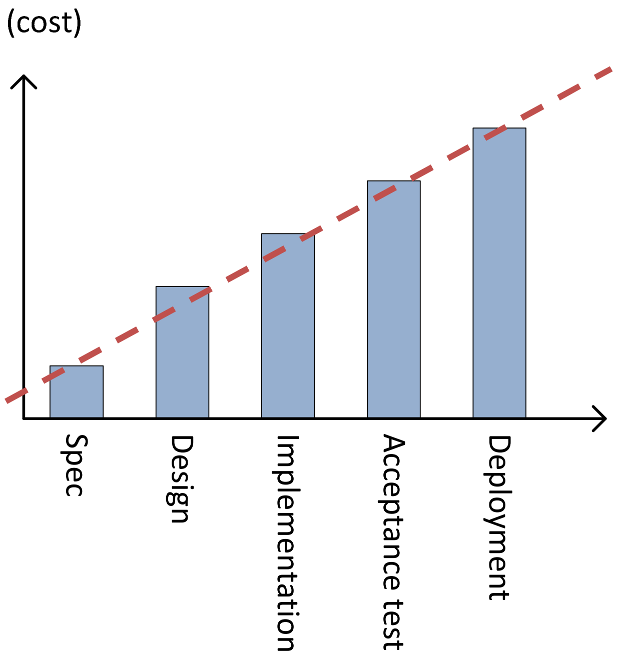
A short discussion

- What is the *value* of testing?
  - For the system
  - For the developer
  - For the company
  - For the customer
  - For the users

- What is the *cost* of testing?

The cost of errors

- Finding errors early is in the best interest of you and your company



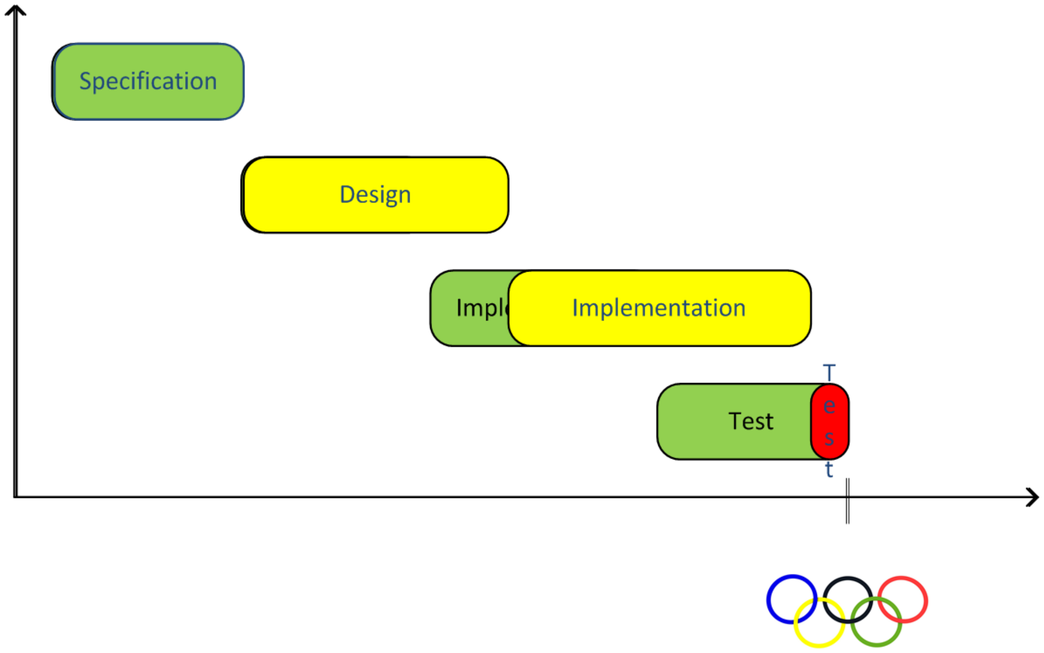
To this, add damage done to

- humans
- property
- company image
- loss of productivity
- follow-on sales

The test mantra:
Test early, test often,
test enough

When to test?

The nightmare, all-too-often-seen scenario



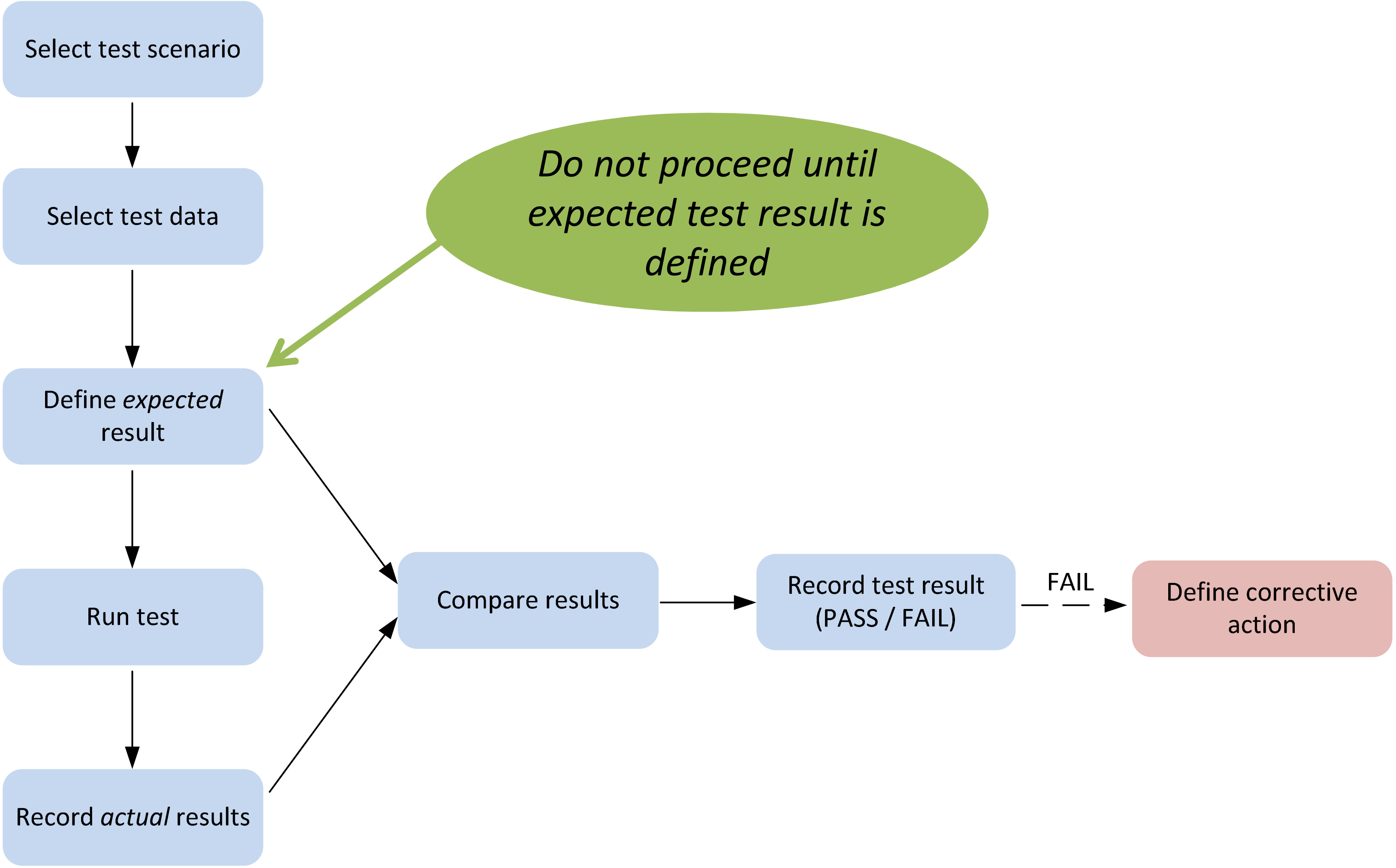
- What happens to the test effort in this case?

Properties of a good test

- What are the properties of a good, valuable test?

- The test should be
  - independent
  - simple
  - repeatable
  - fine-grained
  - quick to run

Defining a test



Selecting test data and Equivalence Classes

- Definition:
  - An Equivalence Class is a collection of input that should be processed and react equally.
- Characteristic:
  - All elements in an equivalence class will either fall or pass.
  - Is used to reduce the number of tests
- Limitation:
  - May require knowledge of: type of processor, programming languages or algorithms

Simple Example

bool Big(int x)

- x can assumed a value in the 1 - 100 range
- if x < 10
  - x is ”small” => false
- else
  - x is ”big” => true

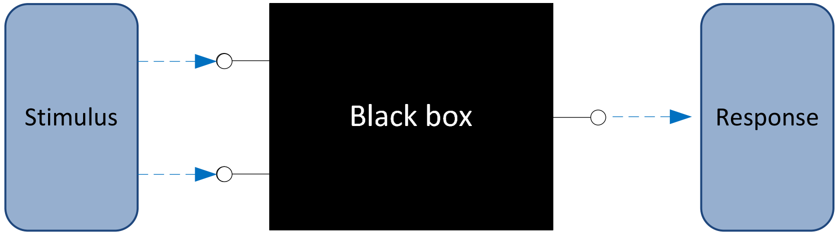
At least 4 Equivalence Classes:

- x < 1 : invalid, but possible input.
- 1<=x<10 : valid data, 1, 5 and 9 chosen as test data.
- 10<=x<100 : valid data, chosen values 10, 49 and 99.
  - Where 10 and 99 is boundary values
- 100 < x : invalid, but possible input.

Test types: Black vs. white box testing

- Black box testing, AKA *functional* testing
  - Test only through system interfaces
  - No knowledge of internal workings

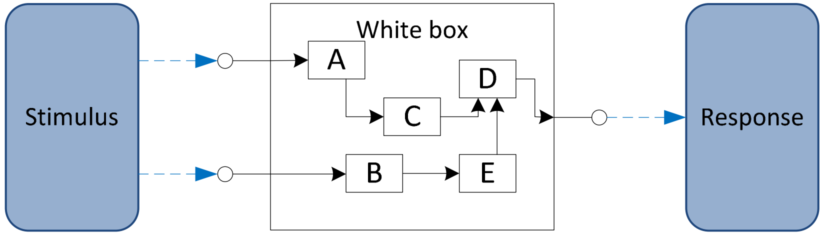
- Complete test → complete set of input tested (valid and invalid)



Test types: Black vs. white box testing

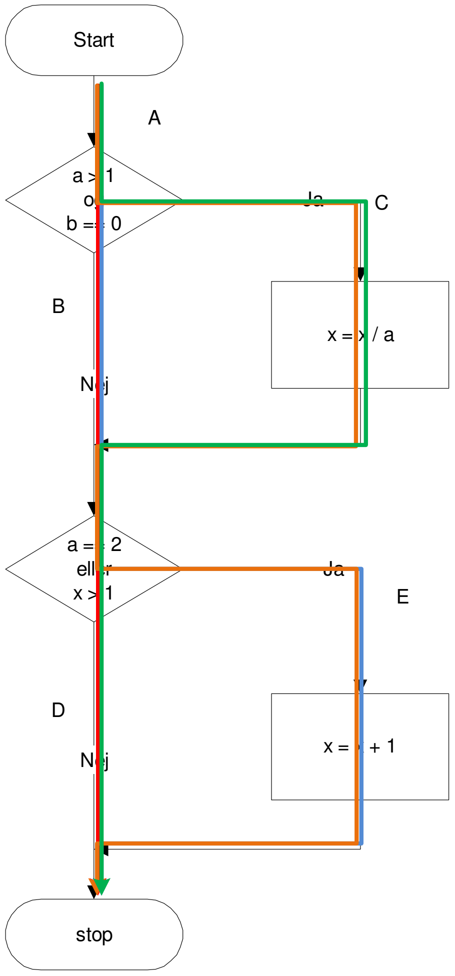
- White box testing
  - Test through system interfaces, but *with* knowledge of internal workings

- Complete test → complete *route coverage*

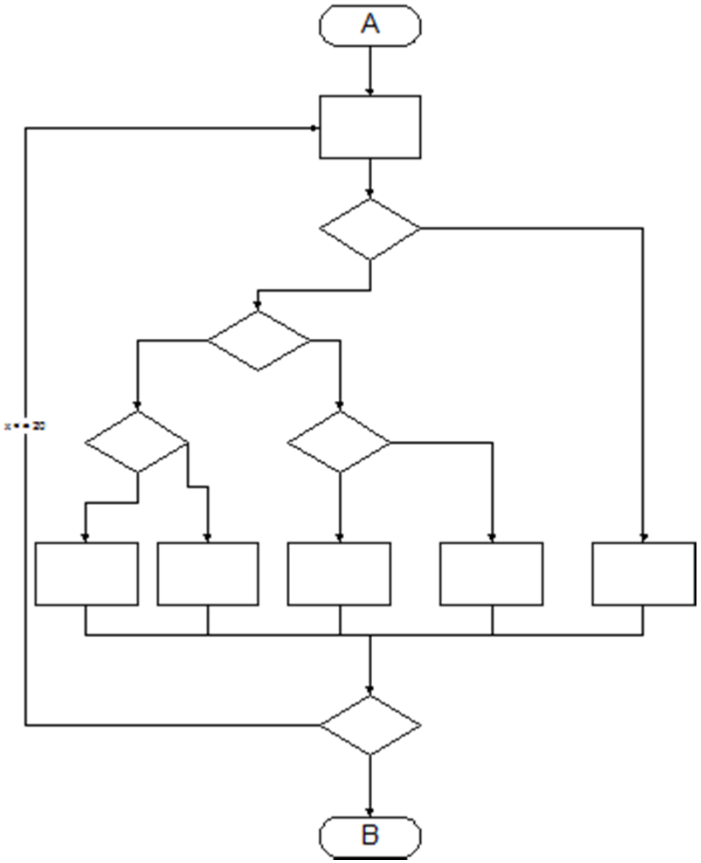


Route coverage - example

```
void f(a, b, x)
{
  if ((a > 1) && (b == 0))
    x = x / a;
  if ((a == 2) || (x > 1))
    x = x + 1 ;
}
```



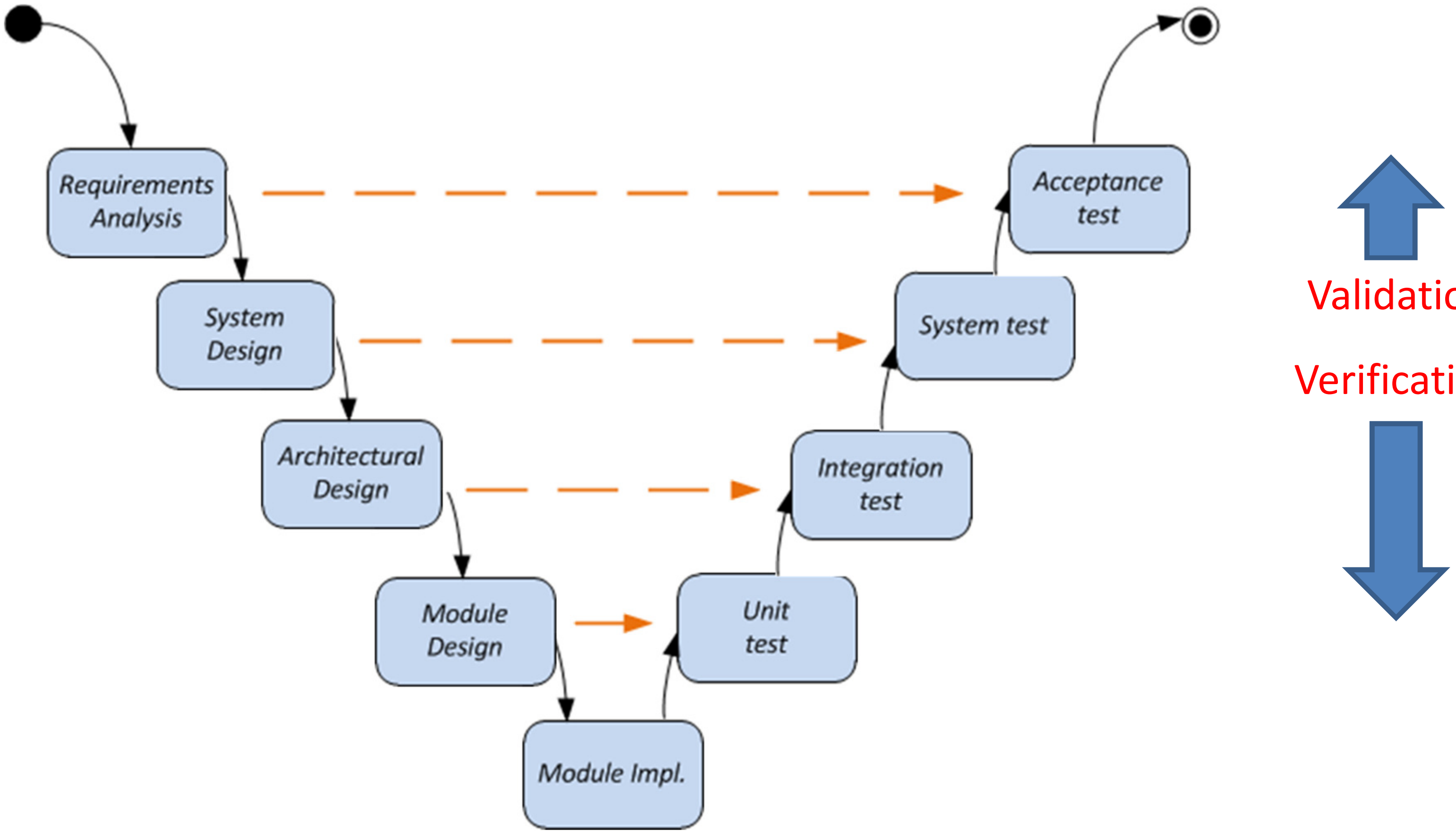
Route coverage - example



- 5 routes, up to 20 loops
- Independent decisions
  → 10^{14} routes

- 1 us/test → 3.17 years

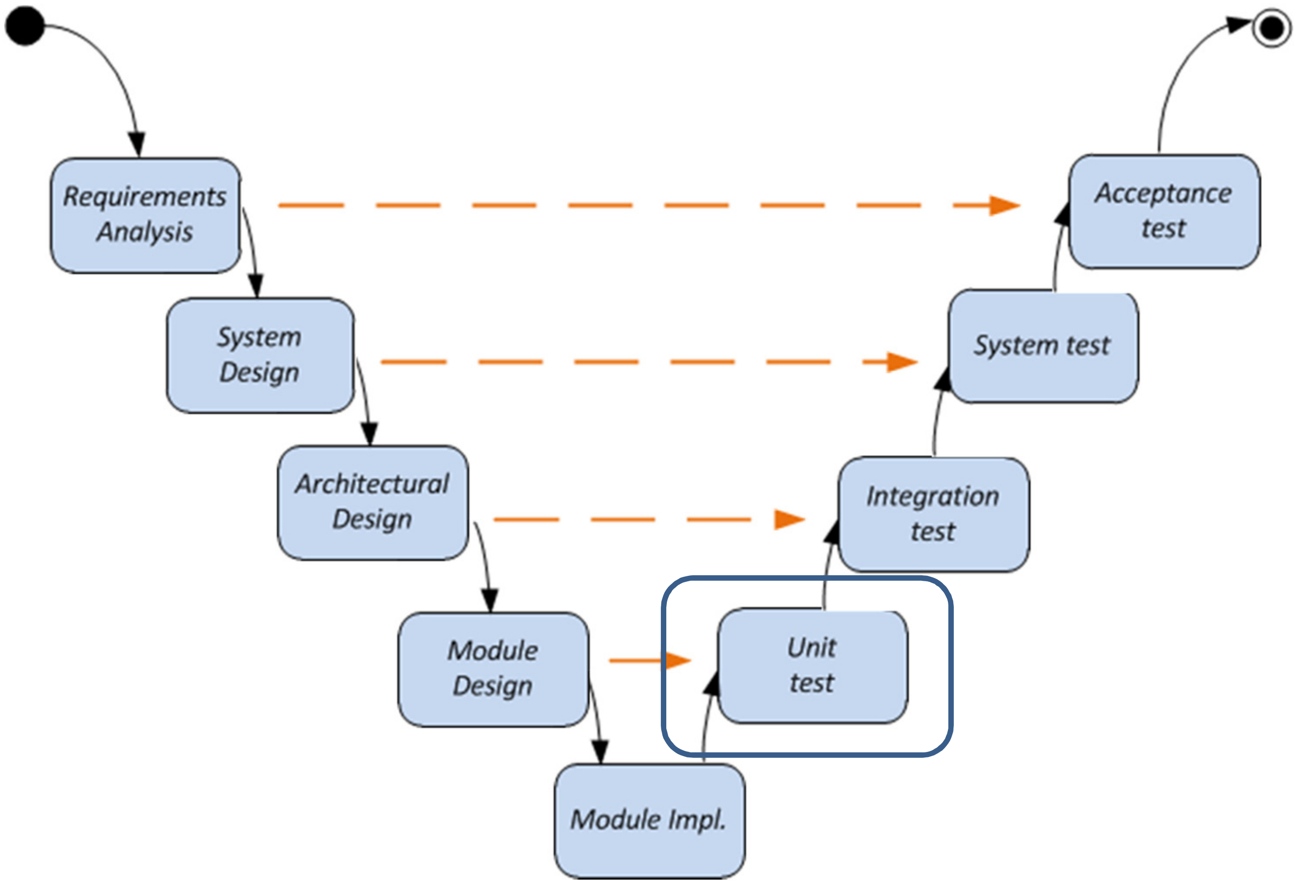
V-Model and Test levels



Validation & Verification

- Validation
  - Build the right thing
  - Checking and testing the product against the requirements and user needs
  - At the end of the development process
  - External process (**System + Acceptance Test**)
- Verification
  - Build the thing right.
  - Complies with a sub requirements regulation, design principles
  - At any given development phase
  - Internal Process (***Unit + Integration Test***)

Test levels


Requirements Analysis
System Design
Architectural Design
Module Design
Module Impl.
Unit test
Integration test
System test
Acceptance test

Test levels: Unit test

- Unit testing is *by far* the most efficient bug-squasher

- Find a bug in unit testing ?
  - correct the bug, re-run the test

- Find *same* bug in acceptance testing ?
  - Explain to customer, schedule new test, damage control, correct bug, regression-test system, …

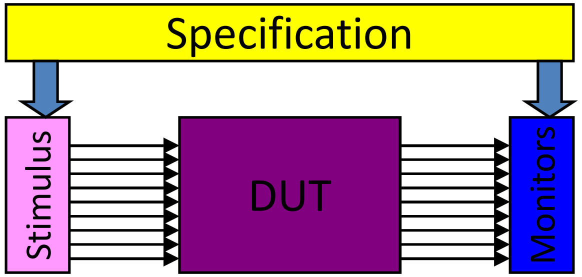
Unit testing in software

- Write software, then write test. Run test, correct bugs, move on…

- Or better yet: Write test, then write software
  - Test Driven Development (TDD)
  - The test becomes a *specification*
  - Red-green-refactor cycle

Unit testing in hardware

- Create component/subsystem, strap to test bench
- Deduct and apply stimulus, observe results
  - Stimuli signals and monitor expected behavior



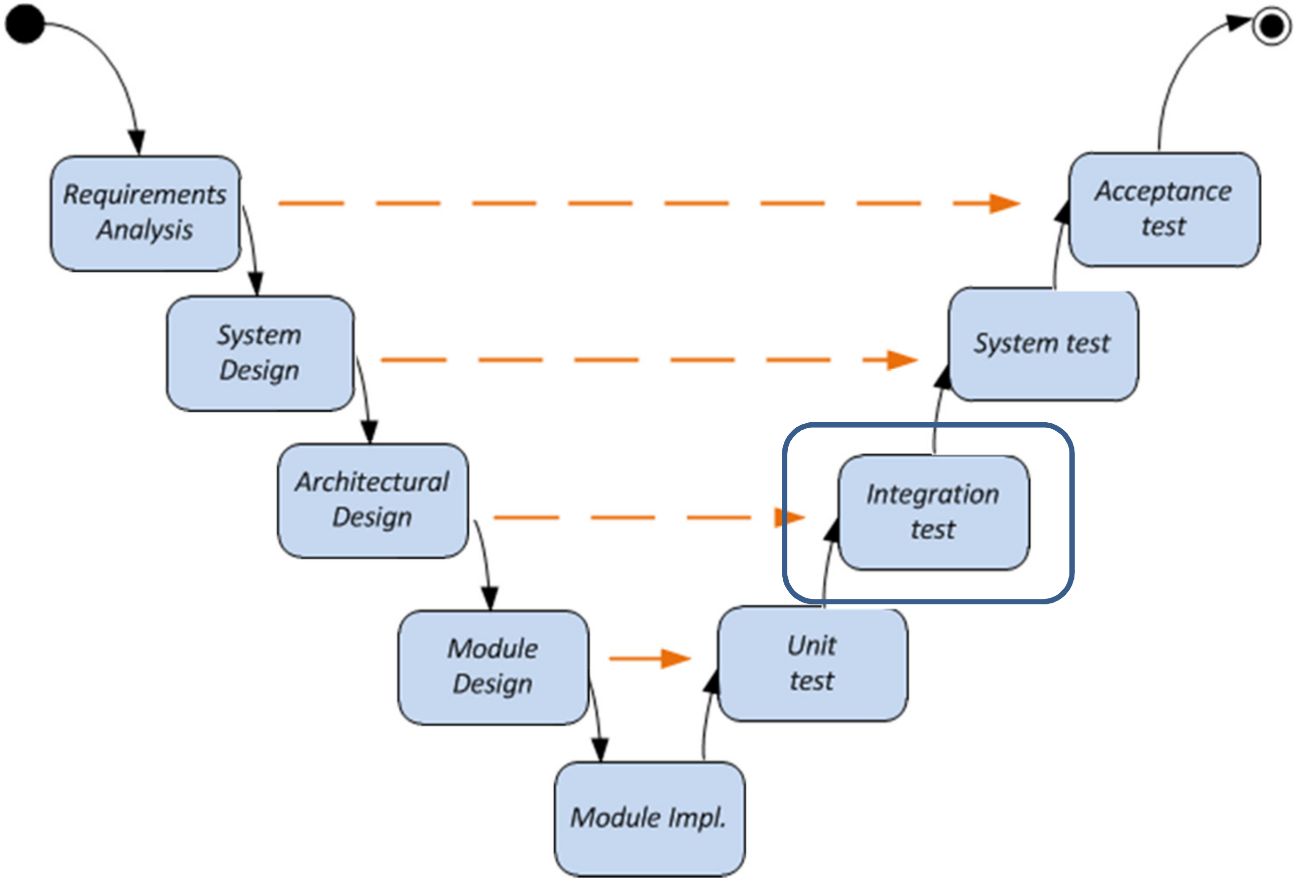
Unit testing

- Unit testing is closely related to design and implementation

- Most often done by implementor – *a problem?*

- Automate tests whenever possible
  - Machines have no feelings

Test levels

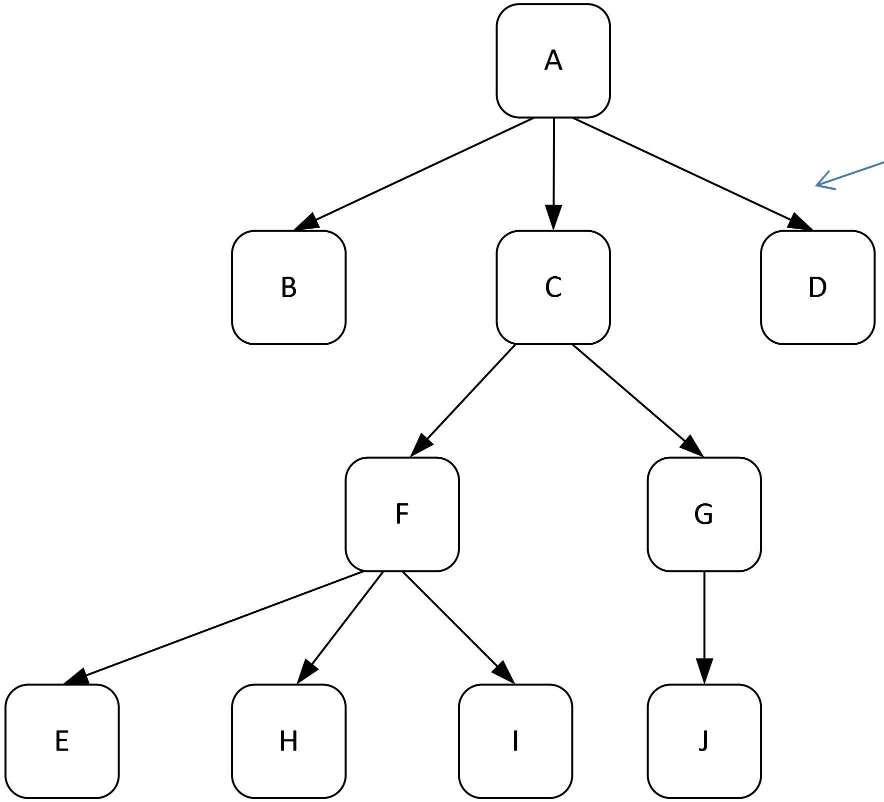

Requirements Analysis
System Design
Architectural Design
Module Design
Module Impl.
Unit test
Integration test
System test
Acceptance test

Test levels: Integration test

- Integration test: Integrating dependent (unit-tested) components

- Various strategies:
  - Big-bang
  - Bottom-up
  - Top-down
  - Sandwich (other hybrids)

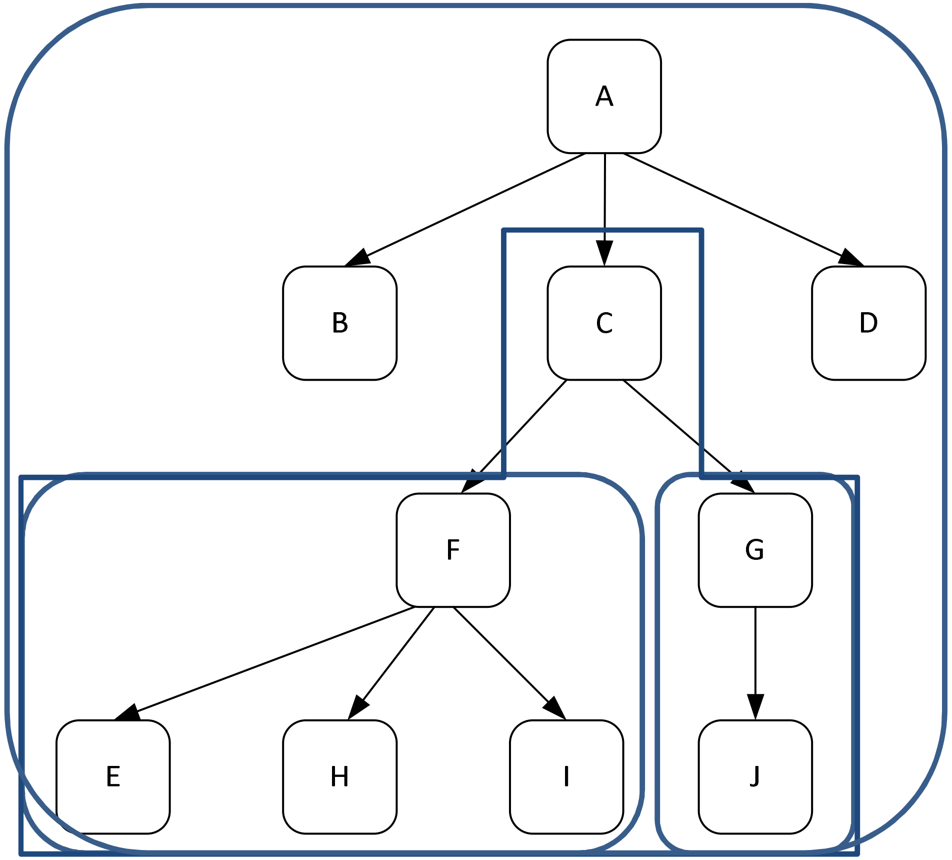
Integration test: Mapping dependencies



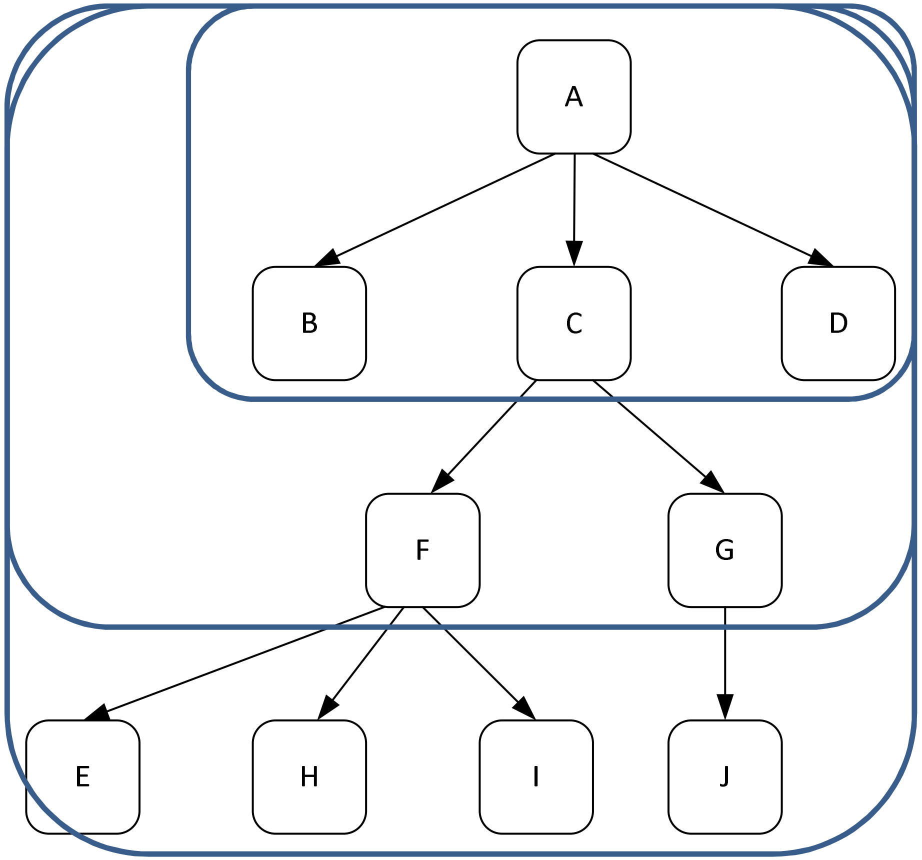
Dependency tree

Integration test:

Bottom-up –
requires *drivers*



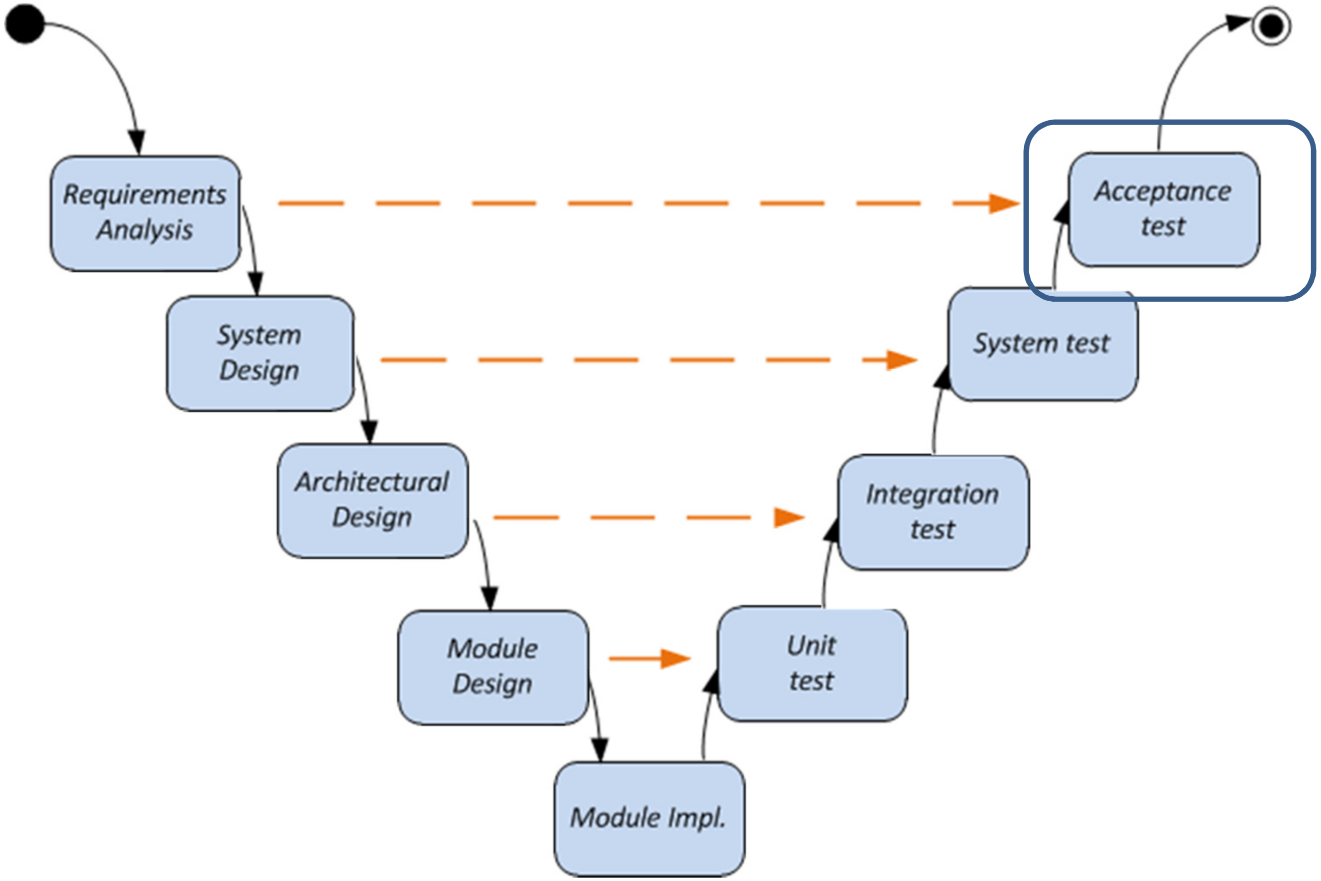
Top-down
- requires *stubs*



Integration test: Discuss

- What are the benefits of top-down integration testing?
- What are the benefits of bottom-up integration testing?
- What is applicable when?
- Do we have to make a one-or-the-other choice?

Test levels


Requirements Analysis
System Design
Architectural Design
Module Design
Module Impl.
Unit test
Integration test
System test
Acceptance test

Acceptance test: UCs versus test

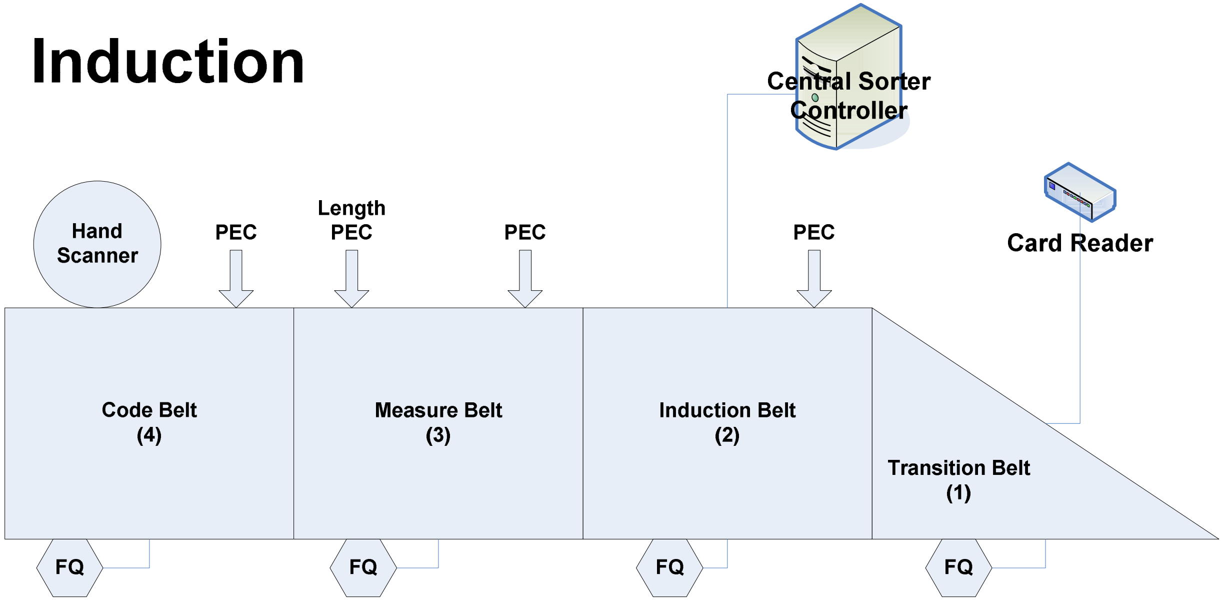
- Conducted with customer – signs off.
- The Use cases (UCs) for the system must map to the acceptance test – why?
- How do we make this happen?

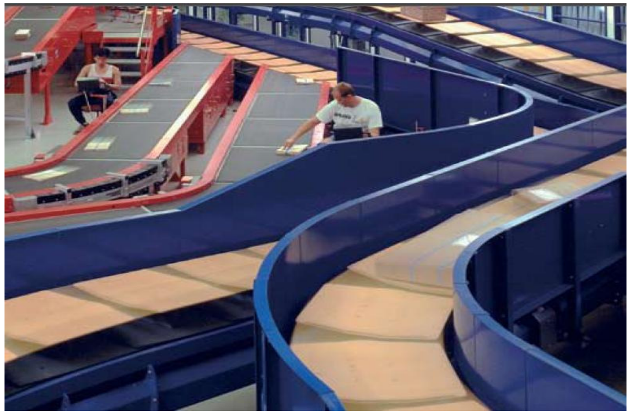
Mapping Use Cases to Acceptance test

- Essentially, performs the use case
  - Scenario maps to steps in the Test
  - Repeatable, because of pre-defined:
    - Input, Output, flow
- Remember:
  - Pre conditions; are they valid?
  - Post conditions; do they hold?
  - A use case may describe multiple paths
    - *For each path you need a test scenario*
- Used to validate the use case

Exercise: **AcceptTestOvelse.pdf**

Exercise – System Test (Black box)





- Find test scenarios
- Find possible test objects
- Think about error scenarios
- Equivalence classes

https://www.youtube.com/watch?v=neSLiifHEBM
https://www.youtube.com/watch?v=tprsTfIRUII