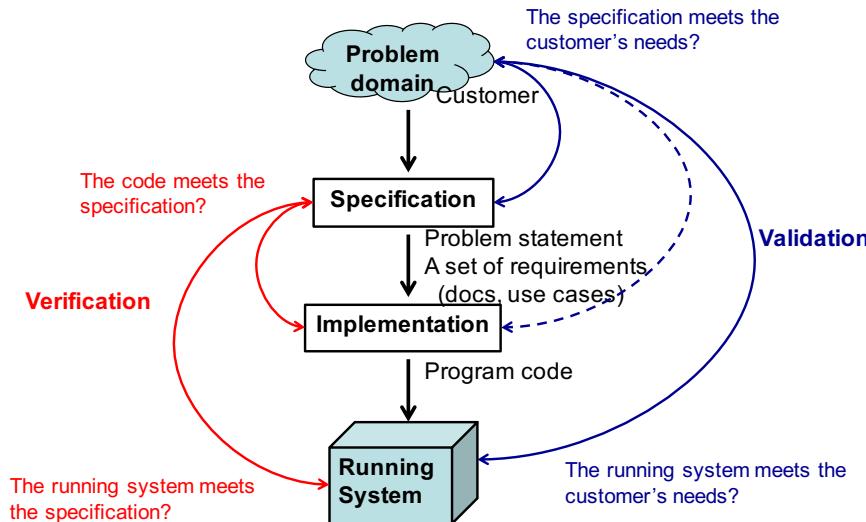


# Software Testing: Verification and Validation

## Verification and Validation (V&V)

- Verification
  - Testing whether a system is developed in accordance with its specification (i.e., a set of requirements).
    - Ensures you built it right.
- Validation
  - Testing whether a system meets your customer's needs.
    - Ensures you built the right thing.

2



- Defects found in verification
  - Occur when the implementation and/or running system fail to meet the specification.
    - The specification of a printer's firmware states that the printer stops printing when its paper tray becomes empty.
    - However, the firmware does not stop a printer when a tray becomes empty.
- Defects found in validation
  - Occur when the specification is wrong or misses the customer's needs.
    - The firmware specification states nothing about what a printer should do when a tray becomes empty.
      - Therefore, the firmware does not stop a printer when a tray becomes empty.
    - However, the customer wants a printer to stop when a tray is empty.

## Example Defects in Validation

- It is possible to correctly define requirements in a specification correctly, so
  - Developers can clearly tell what features to implement and how to implement those features.
- However, it is hard to make the specification sufficiently comprehensive, so
  - Developers do not miss the customer's requirements.
  - Requires numerous “what-if” questions.
    - What if a tray becomes empty? Printing job should stop? What if another tray has papers? Can a printer still accept print jobs from computers?

5

- Firmware for Boeing 787's generator control unit (GCU)
  - Does periodic “status check” every 10 milliseconds.
  - Implements a counter (timestamp) with signed 32-bit integer.
    - $2^{31} = 2,147,483,648$
    - $10 \text{ msec} * 2147483648 = 248.551 \text{ days}$
    - An integer overflow occurs once GCU operates for 248.551 days.
- GCUs fall into a failsafe mode if they are continuously powered on for 248 days.
  - A 787 aircraft has 4 GCUs.
  - If all of them are powered on at the same time, the aircraft can lose its control completely.

Counter	Status
0	G
1	G
2	G
⋮	⋮



## XXX-day Problems

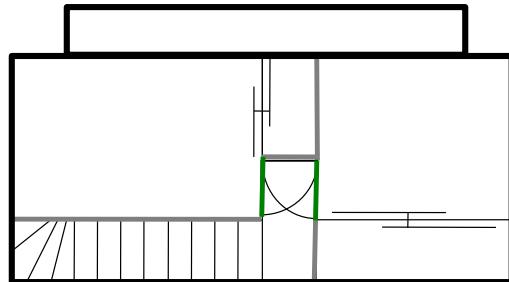
- Customer
  - Didn't think and wasn't asked how long a GCU should be able to keep running if it is not turned off.
    - Status check might look like a minor feature in GCU development.
- Developer
  - Didn't think and wasn't instructed (by the specification) about how long a GCU should be able to keep running if it is not turned off.
  - Decided to use the simplest type for the counter and didn't have a chance to think more about it.
    - Status check might look like a minor feature in firmware development.

- 248-day problem
- 494-day problem
  - Occurs if a counter/timer relies on an unsigned 32-bit integer
    - Server OSes, WiFi routers, network switches, etc. etc.
- 24-day and 49-day problems
  - Occur if a counter/timer relies on a signed 32-bit integer and its counting/timing resolution is 1 msec.
- 830-day problem
  - Occurs if a counter/timer relies on an unsigned 32-bit integer and its counting/timing resolution is 60 Hz (1/60 second; 16.67 msec)
- Year 2038 problem (Unix millennium bug)
  - Many OSes have a timer that counts time in second from 1970/1/0:00:00, using a signed integer. The timer will overflow at January 19 in 2038.

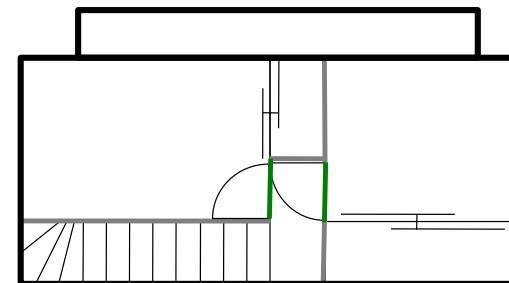
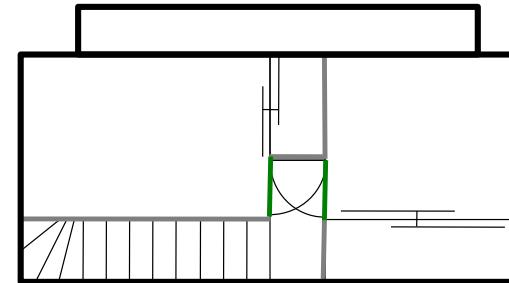
7

8

## When I was a kid...



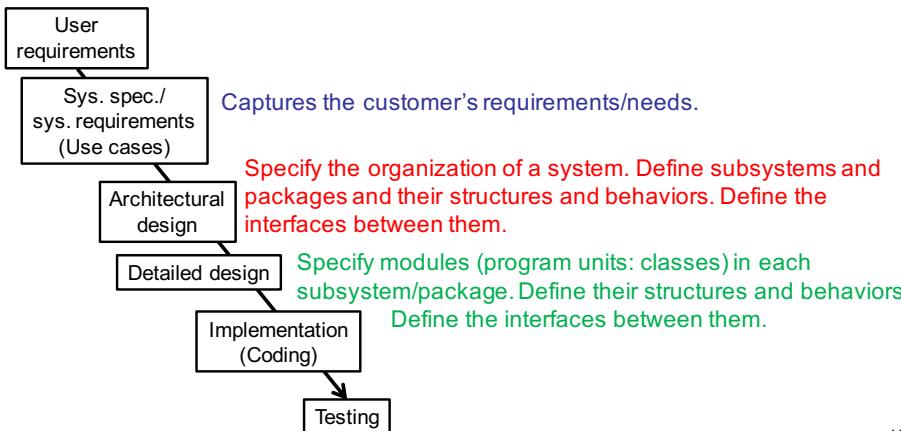
9



10

## Waterfall Process Model

- One of the earliest models to describe development processes.



11

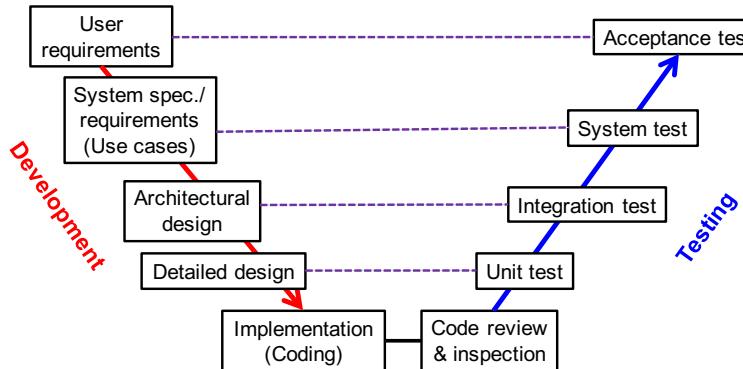
## Problems in Waterfall Process

- Defects are found at the end of the project.
  - Testing does not take place until the end of the project.
- It is often too late and too costly to push feedback up the waterfall.

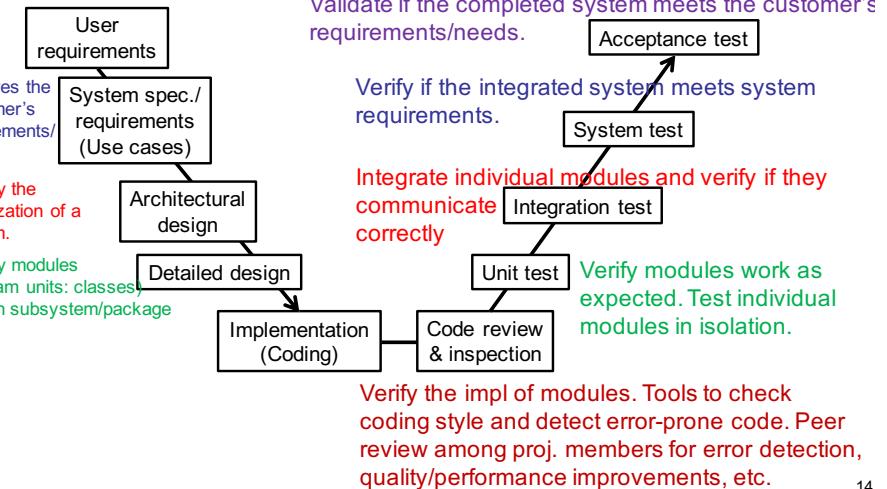
12

# V-Model

- Extends the waterfall model.
  - Testing phase is expanded
- Explicitly states which testing phase corresponds to which development phase.

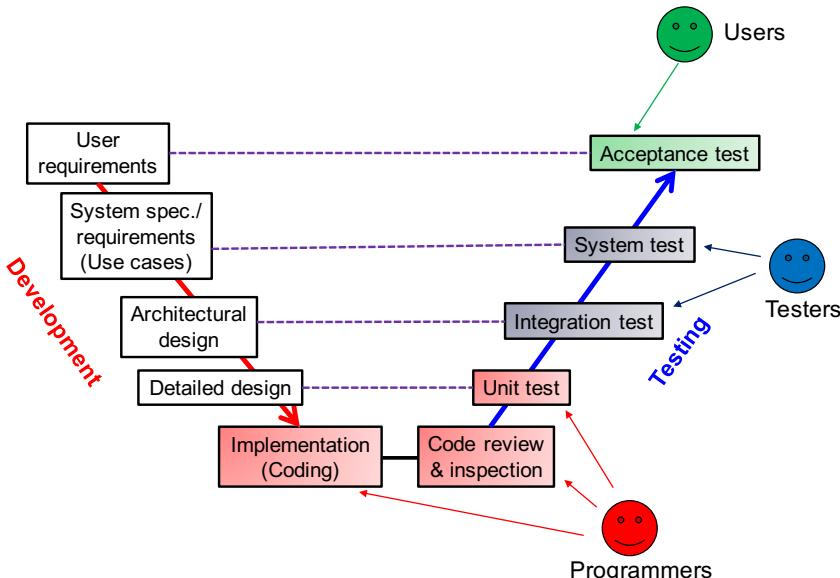


13



14

## Division of Responsibilities



15

## Test Levels and Test Types

- Test level**
  - A group of test activities that are organized and managed together.
  - Linked to the responsibilities in a project.
  - e.g., unit test, integration test, system test and acceptance test.
- Test type**
  - Focuses on a particular test objective.
    - e.g. functional test, non-functional test, structural test, confirmation test, etc.
  - Takes place at one test level or at multiple levels.

16

- Test levels and test types are orthogonal.
- Different projects have different policies on which test types involve in which levels.

	Functional test	Non-functional test	Structural test	Confirmation test
Acceptance test				
System test				
Integration test				
Unit test				
Code rev&insp.				

- For example...

	Functional test	Non-functional test	Structural test	Confirmation test
Acceptance test	X	X		
System test	X	X		X
Integration test	X	?	X	X
Unit test	X	?	X	X
Code rev&insp.	X	?	X	X

17

18

## Test Types: Functional Test

- Functional test
  - Focuses on the behaviors of tested code/software
    - Driven by the descriptions and use cases specified in the specification.
  - Black-box testing
    - Treat the tested code/software as a black-box
      - Testing *without* knowing the internals of tested code/software
    - Give an input to tested code/software and compare its output with the expected result.
      - Coarse-grained testing: Testing the external behaviors of tested code/software

	Functional test	Non-functional test	Structural test	Confirmation test
Acceptance test	X	X		
System test	X	X		X
Integration test	X	?	X	X
Unit test	X	?	X	X
Code rev&insp.	X	?	X	X

19

20

# Test Types: Non-Functional Test

- Non-functional test
  - Focuses on the non-functional quality characteristics of tested code/software.
    - Driven by the descriptions specified in the specification.
  - Security test
    - Check if security vulnerability exists in tested code/software.
  - Usability test
    - Ease of use/browse/comprehension, intuitive page/screen transition
  - Efficiency test
    - Performance (e.g. response time, throughput), resource utilization (e.g. memory, disk, bandwidth)
  - Environmental test
    - Configuration/compatibility test
      - Can the tested code/software be installed on certain OS(es) and HW(s)?
      - How does the tested code/software behave on certain OS(es) and HW(s)?
      - How does the tested code/software interact with an external required service(s)?
        - » Does it work with Version X of the service? How about Version Y?
    - Co-existence test
      - Can the tested code/software run correctly when other software/services run on the same machine?

21

## – Reliability test

- Stress test (load test)
  - How does tested code/software behave under an excessive load?
    - » Example loads: huge data inputs, numerous network connections
- Long-run test
  - Does performance degrade when tested code/software runs for a long time?
- High frequency test
  - How does tested code/software behave when it repeats a certain task at excessively high frequency?
- Fault-tolerance test
  - Can a tested code/software continue its operation under a fault?
- Recoverability test
  - How can a tested code/software recover its operation and data after a disaster (e.g. physical damages of hardware, blackout)?
- Compliance test
  - Data retention, access control, logging, etc.

22

	Functional test	Non-functional test	Structural test	Confirmation test
Acceptance test	X	X		
System test	X	X		X
Integration test	X	?	X	X
Unit test	X	?	X	X
Code rev&insp.	X	?	X	X

23

24

# Test Types: Structural Test

- Testing the structure of an individual module or a set of (integrated) modules.
  - The structure of a system (i.e. a full set of (integrated) modules) = system architecture
- Revise the structure, if necessary, to improve maintainability, flexibility and extensibility.
  - Refactoring
    - » e.g. Replacing conditionals with polymorphism, replacing a magic number with a symbolic constant.
    - » Revising the interfaces of modules if an integration test fails.
    - » Interface: How modules interact with each other
  - Use of design patterns
    - » e.g., Replacing conditionals with the *State* design pattern
- White-box testing
  - Treat tested code/software as a white-box
    - Testing *with* the knowledge about the internals of the tested code/software
  - Fine-grained testing: Taking care of internal behaviors of tested code/software

	Functional test	Non-functional test	Structural test	Confirmation test
Acceptance test	X	X		
System test	X	X		X
Integration test	X	?	X	X
Unit test	X	?	X	X
Code rev&insp.	X	?	X	X

25

26

# Test Types: Confirmation Test

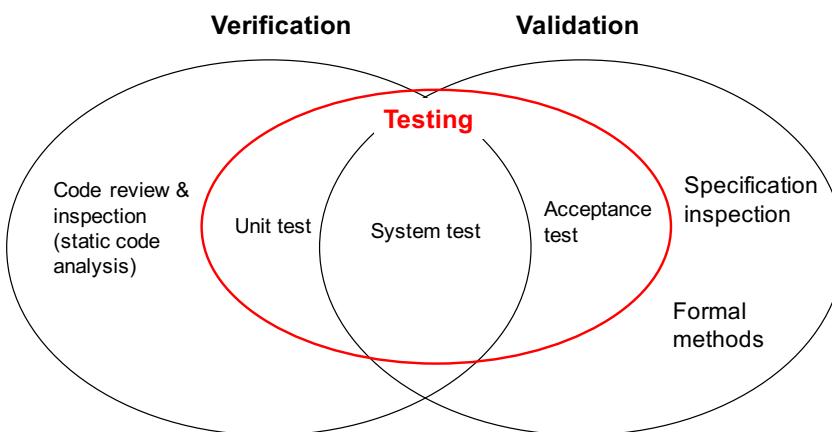
- Re-testing
  - When a test fails, detect a defect and fix it. Then, execute the test again
    - To confirm that the defect has been fixed.
- Regression testing
  - In addition to re-testing, execute ALL tests to check that the tested code/software has not regressed.
    - That is, it does not have extra defects as a result of fixing a bug.
    - Verifying that a change in the code/software have not caused unintended negative side-effects and it still meets the specification.

	Functional test	Non-functional test	Structural test	Confirmation test
Acceptance test	X	X		
System test	X	X		X
Integration test	X	?	X	X
Unit test	X	?	X	X
Code rev&insp.	X	?	X	X

27

28

## V/V Methods

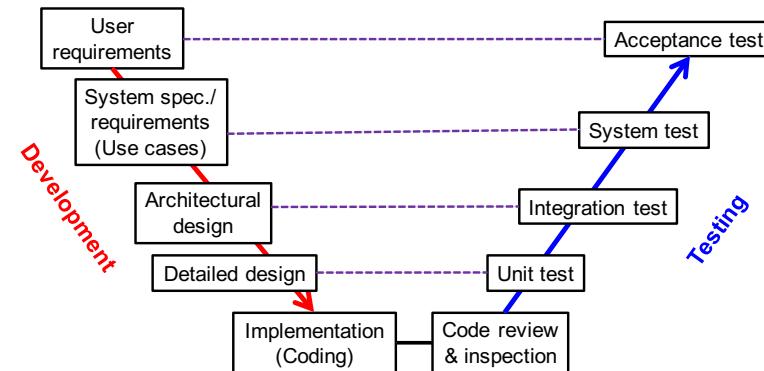


29

## Unit Testing

## V-Model and Development Process

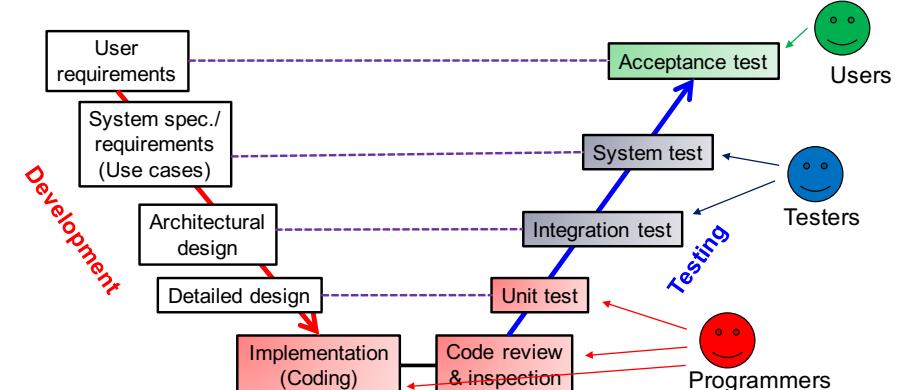
- V-Model can be used to define various iterative development process,
  - beyond waterfall process



30

## Unit Tests

- Verify that each *program “unit”* works as it is intended and expected along with the system specification.
  - Units to be tested: each class/interface (a set of methods) in OOPs



31

32

## Who Does it?

- You as a programmer do it.
- Test cases are written as programs from a programmer's perspective.
  - A test case describes a test to verify tested module/class against a requirement.
- Programmers and unit testers are no longer separated in most (both large-scale and small-scale) projects as
  - it has been a lot easier and less time-consuming to write and run unit tests.
  - programmers can write the best test cases for their own code in the least amount of time.

33

## What to Do in Unit Testing?

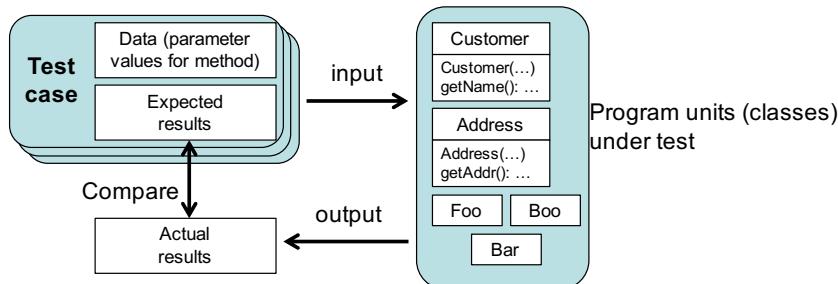
- 4 tests (test types)
  - CS680 focuses on 3 of them: *functional*, *structural* and *confirmation* tests.

	Functional test	Non-functional test	Structural test	Confirmation test
Acceptance test				
System test				
Integration test				
<b>Unit test</b>	<b>X (B-box)</b>	<b>?</b>	<b>X (W-box)</b>	<b>X</b>
Code rev&insp.				

34

## Functional Test in Unit Testing

- Ensure that each method of a class successfully performs a set of specific tasks.
  - Each test case confirms that a method produces the expected output when given a known input.
    - Black-box test
  - Well-known techniques: equivalence test, boundary value test



35

## Structural Test in Unit Testing

- Verify the structure of each class.
- Revise the structure, if necessary, to improve maintainability, flexibility and extensibility.
  - White-box test
- To-dos
  - Refactoring
  - Use of design pattern
  - Control flow test
  - Data flow test

36

## Confirmation Test in Unit Testing

- Re-testing
- Regression testing

## HW: Reading Assignment

- Nick Jenkins, *A Software Testing Primer: An Introduction to Software Testing*, 2008.
  - <http://www.nickjenkins.net/prose/testingPrimer.pdf>