Software Verification & Validation

Natthapong Jungteerapanich

Handout 1

Acknowledgement

- Some slides in this lecture are adapted from
  - Slides from **Dr. Jittat Fakcharoenphol**'s Software Verification and Validation course at Kasetsart University
  - Slides from **Prof. George C. Necula**'s Software Engineering course at Stanford University
  - **Ammann and Offutt's Introduction to Software Testing**, Cambridge University Press, 2008

Software Verification and Validation

- Definitions
  - **Software Verification** is the process of checking that the software is built correctly according to its specification
  - **Software Validation** is the process that ensures that the software meets the customer's expectation

- In short,
  - **Software Verifier** asks *"Are we building the software right?"*
  - **Software Validator** asks *"Are we building the right software?"*

Two Main Methods of V&V

In this course, we will look at two main methods for software V&V

1. Software Testing

2. Formal verification

The Adriene Rocket Disaster

- On June 4, 1996 an unmanned Ariane 5 Flight 501 rocket launched by the European Space Agency exploded 37 seconds after its lift-off. The rocket was on its first voyage after decade of development costing $7 billion. The destroyed rocket and its cargo were valued at more that $500 million.

Picture from http://www.ima.umn.edu/~arnold/disasters/ariane.html

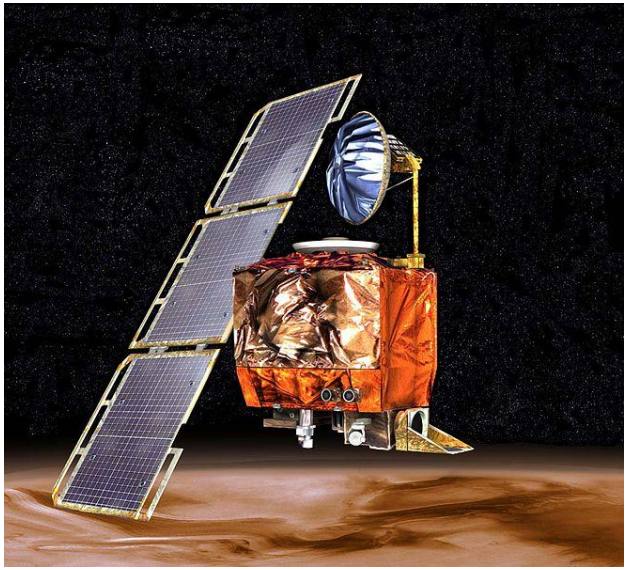
The Adriene Rocket Disaster

- Araine 5 reused the flight control software software from Ariane 4. But the Araine 5 flight path was considerably different and beyond the specification of Ariane 4 software.

- **Cause:** A data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This created an *exception,* but the handler for this exception was not well designed and written.

Mars Climate Orbiter

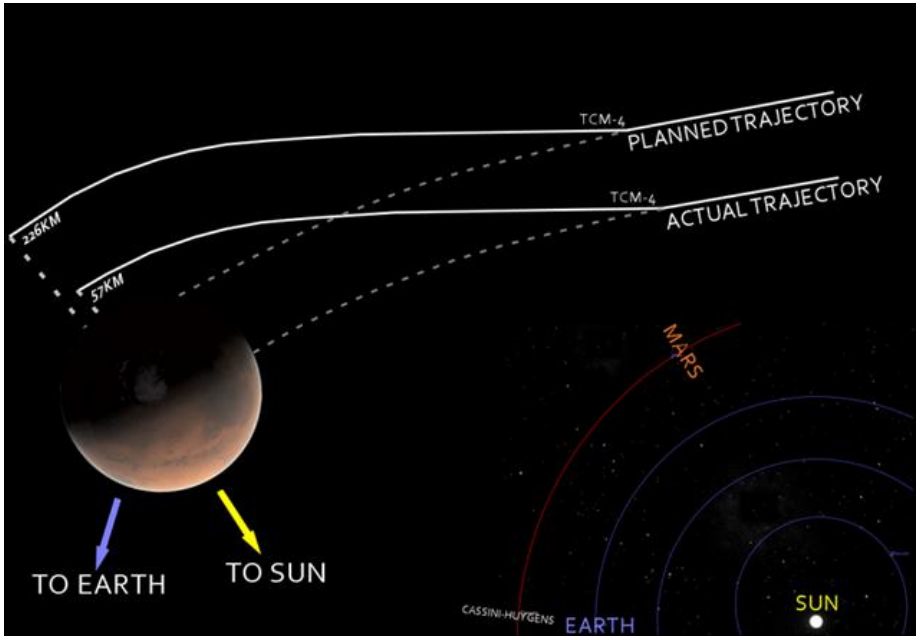
- Mars Climate Orbiter was a robotic space probe launched by NASA on December 11, 1998 to study the Martian climate and acts as a communications relay for Mars Polar Lander.
- The spacecraft entered Mars at an improperly low altitude, causing it to incorrectly enter the atmosphere and disintegrate.



Picture from Wikipedia

Mars Climate Orbiter

- *Mars Climate Order* was built by Lockheed Martin. NASA Jet Propulsion Lab was responsible for the flight.
- **Cause:** The flight system software on the *Mars Climate Orbiter* was written to calculate thruster performance using the metric unit *Newtons (N),* while the NASA team entered data using the Imperial unit *Pound-force (lbf)*.
- **Total Loss:** > $100 million



Testing

- **Testing** is the process of finding faults in the software.

- Testing can help increase the confidence that the software is correct against the specification.

- However, testing **cannot** 100% guarantee that the program is correct.

Faults, Error, and Failure

- **Definitions**
  - **Software Fault**: a defect in the software, also called a **bug**
  - **Software Error**: an incorrect internal state caused by some fault
  - **Software Failure**: Incorrect behaviors when the software runs

Medical Analogy

Got a headache and a sore throat = **Failure**

Physical Examination

High body temperature and inflammation of the throat = **Error**

Diagnosis

Common cold virus = **Fault**

The First Bug

9/9

0800 Arctan started { 1.2700 9.037 847 025
1000 " stopped - arctan ✓ 9.037 846 ?95 correct
13"c (032) MP - MC ~~1.6821[illegible]000~~ ~~2.13047[illegible]15 (-3)~~ 4.615925059(-2)
(033) PRO 2 2.130476415
correct 2.130676415
Relays 6-2 in 033 failed special speed test
In relay " 11.00 test.
Relays changed
1100 Started Cosine Tape (Sine check)
1525 Started Multy Adder Test.

1545 Relay #70 Panel F
(moth) in relay.
First actual case of bug being found.
1630 Arctangent started.
1700 closed down.

Faults, Error, and Failure

```
public static int numZero(int[] x) {
    // Effects: if x == null throw NullPointerException
    //          else return the number of occurrences of 0 in x

    int count = 0;
    for(int i = 1; i < x.length; i++) {
       if (x[i] == 0)  count++;
    }
    return count;
}
```

Test case 1: `numZero([2, 7, 0])` **Expected output** = 1

Test case 2: `numZero([0, 7, 2])` **Expected output** = 1

Testing and Debugging

- **Definitions**
  - **Testing:** evaluates the software by observing its execution
  - **Test failure:** a failure observed during testing
  - **Debugging:** identifies the faults which cause the test failure and correct them

- Testing may not trigger a fault into an incorrect behavior (i.e. a failure).

- Conditions for a fault to create a failure during a test:
  - The location in the program that contain the fault must be reached.
  - After executing the location, the state of the program must be incorrect (i.e. error must occur).
  - The incorrect state must create observable incorrect output or behavior.

Testing in the SDLC

- The V model of software development processes and testing levels.

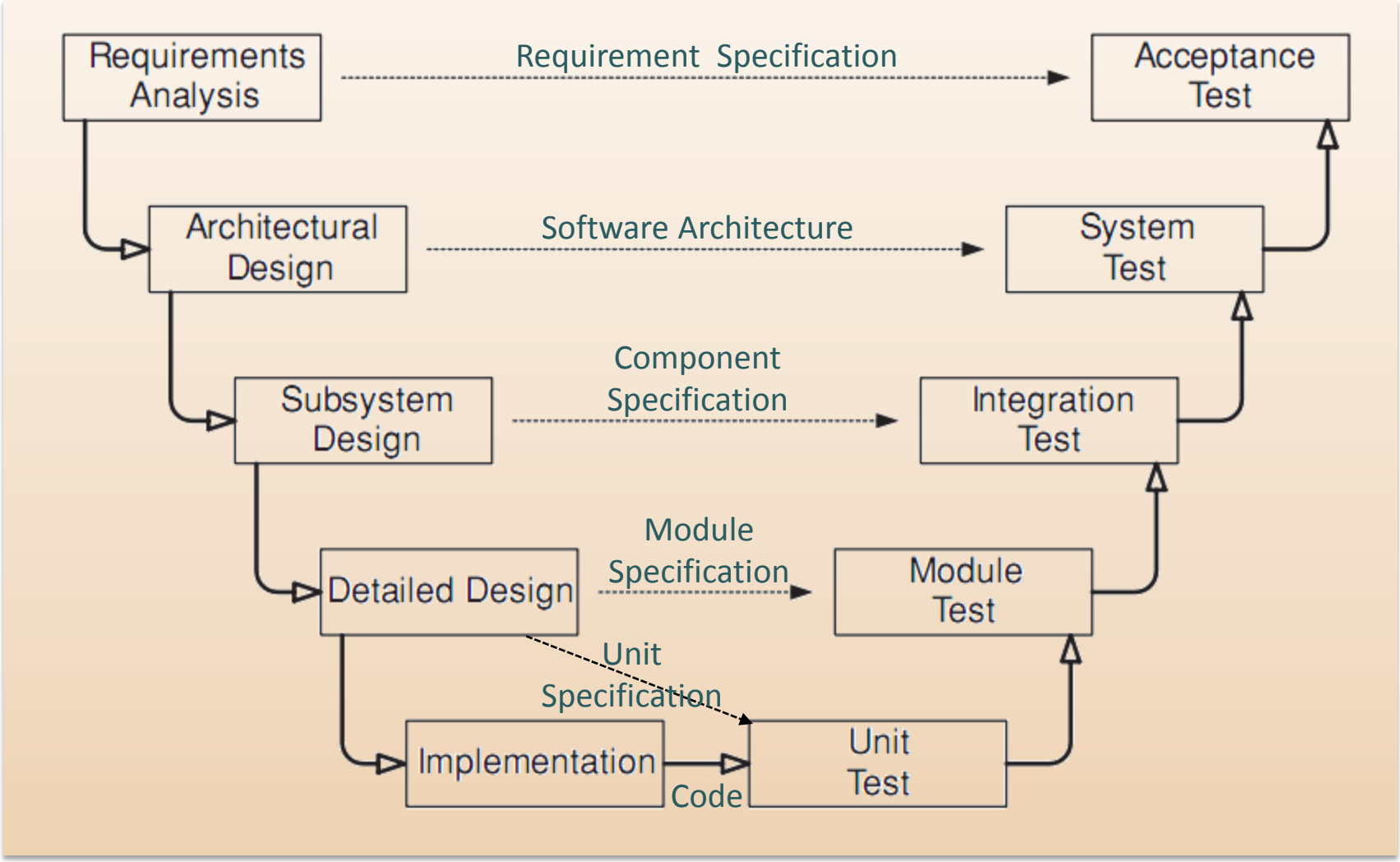


Figure from Ammann and Offutt's Introduction to Software Testing

Levels of Testing

- **Requirement analysis phase:** produces the requirement specification which captures the customer's needs.
  - **Acceptance testing**: check whether the completed system meets those needs. Involves customers or users with domain knowledge.

- **Architectural design phase**: designs which components (SW and/or HW) and how they are connected to obtain the system satisfying the specification.
  - **System testing**: check that the system as a whole meets the specification, assuming that each individual component works correctly.

Levels of Testing

- **Subsystem design phase**: designs the structure and behavior of each component to satisfy the overall architecture
  - **Integration testing**: checks that the modules in the software component work together correctly, assuming that each individual module works correctly.

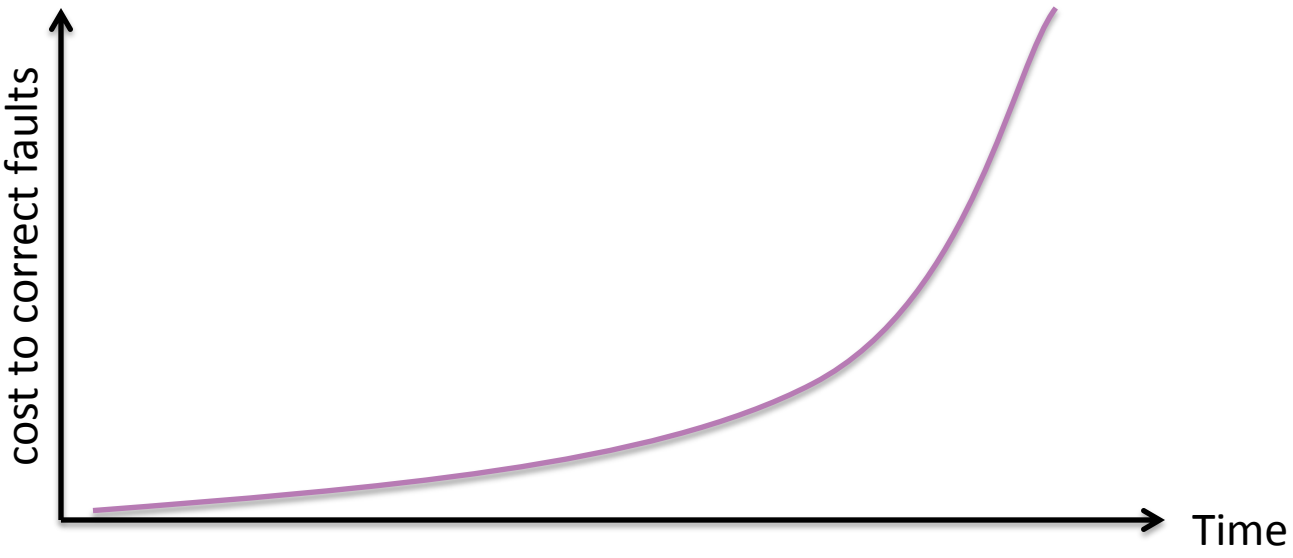
- **Detailed design phase:** designs the structure and behavior of each module
  - **Module testing:** check that each individual module works as expected, assuming that each individual unit in the modules works correctly.

Levels of Testing

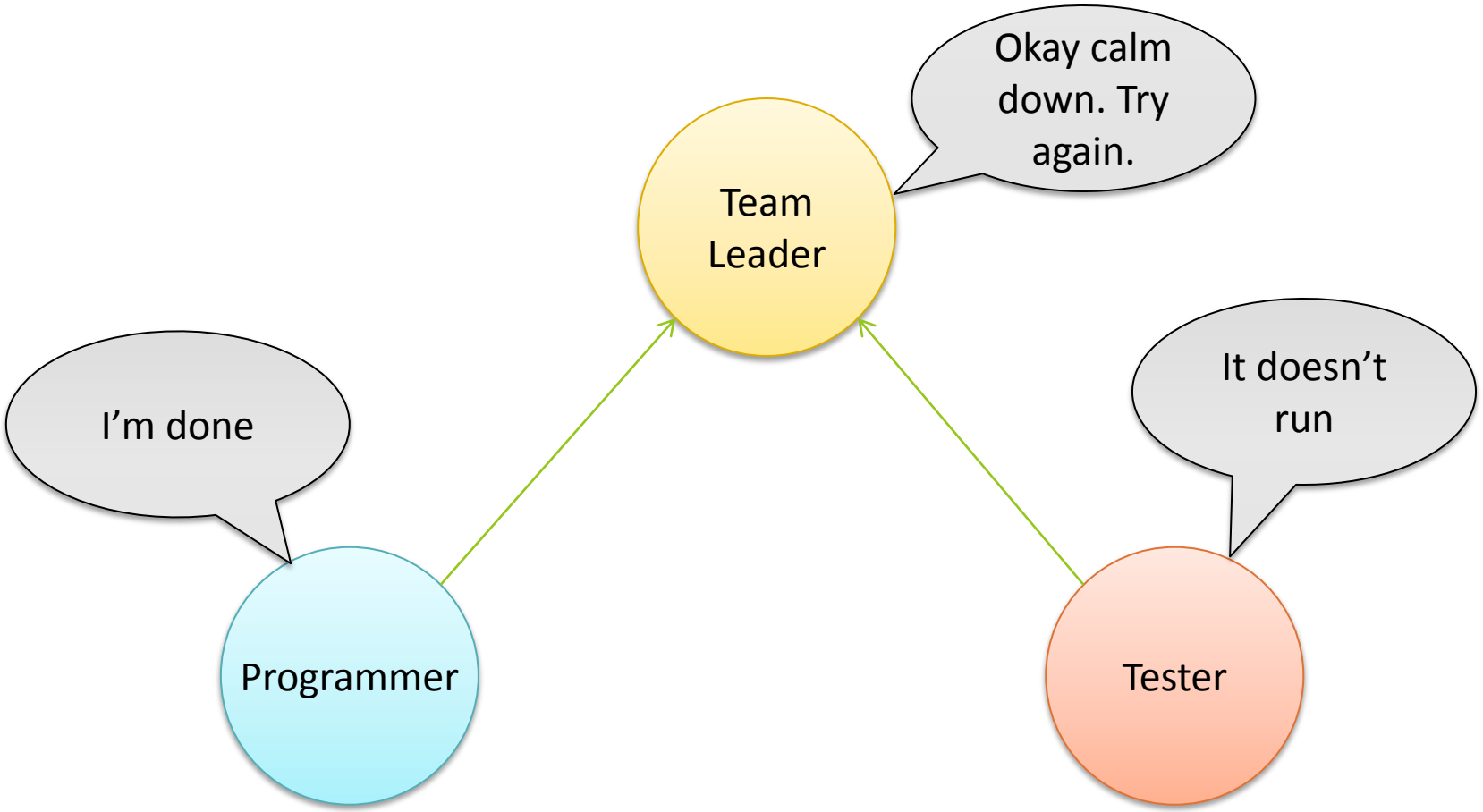
- **Implementation phase**: produces code according to design specification
  - **Unit testing**: checks that each unit behaves as expected. The lowest level of testing.

Test early and often

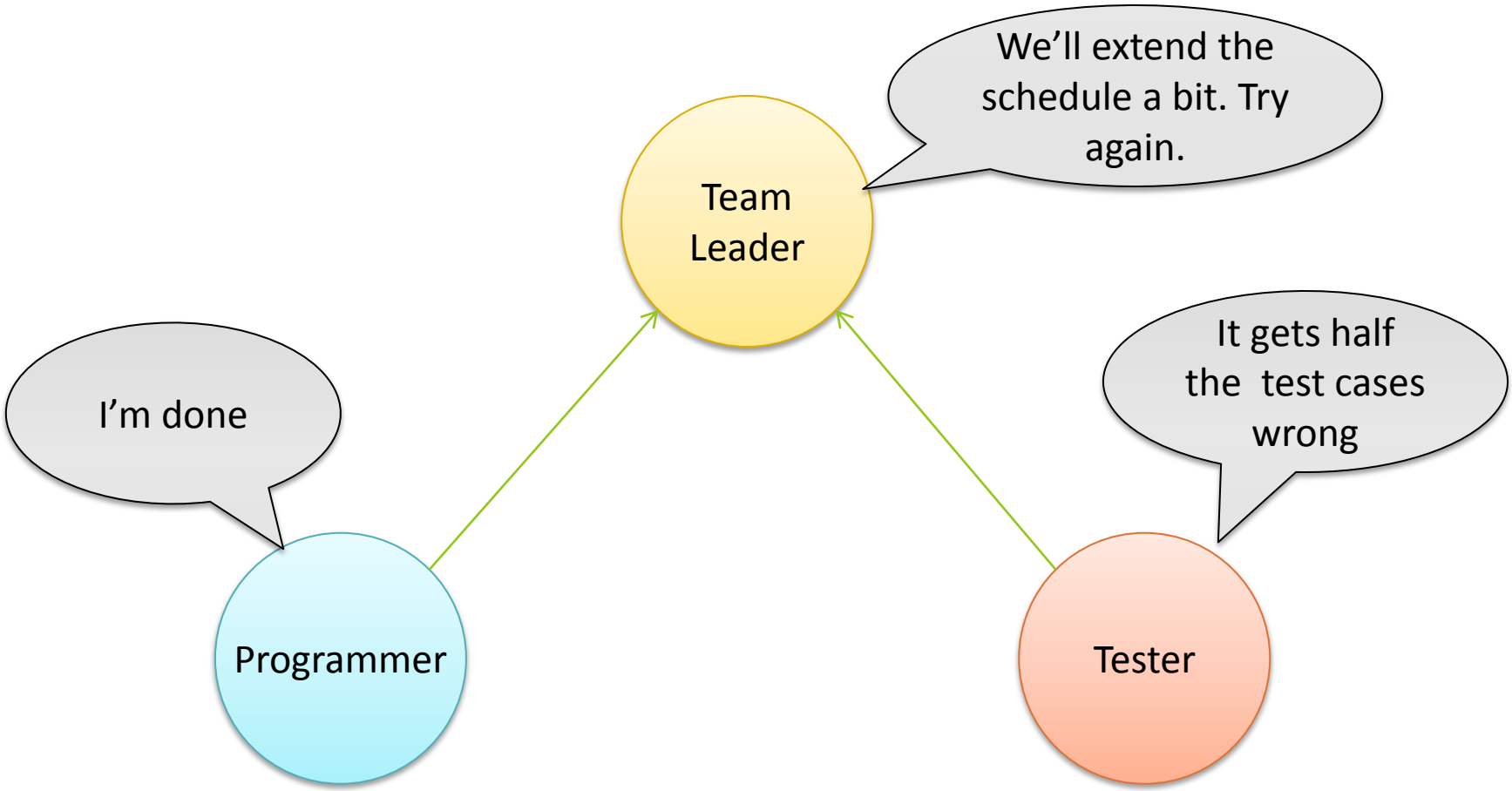
- Testing should be done at all **stages** in the SDLC and **as early and often as** possible.



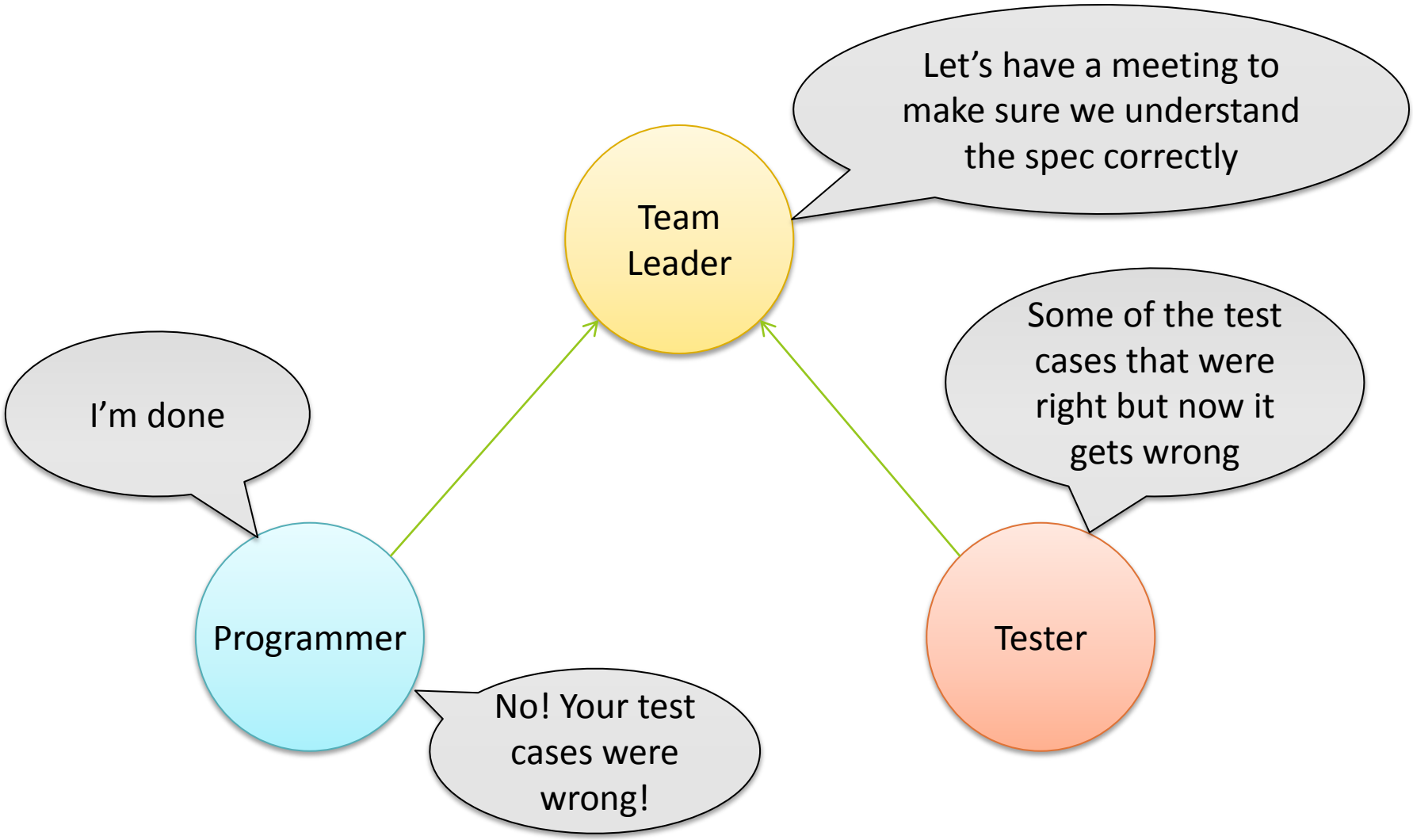
Problem of testing only at the last stage



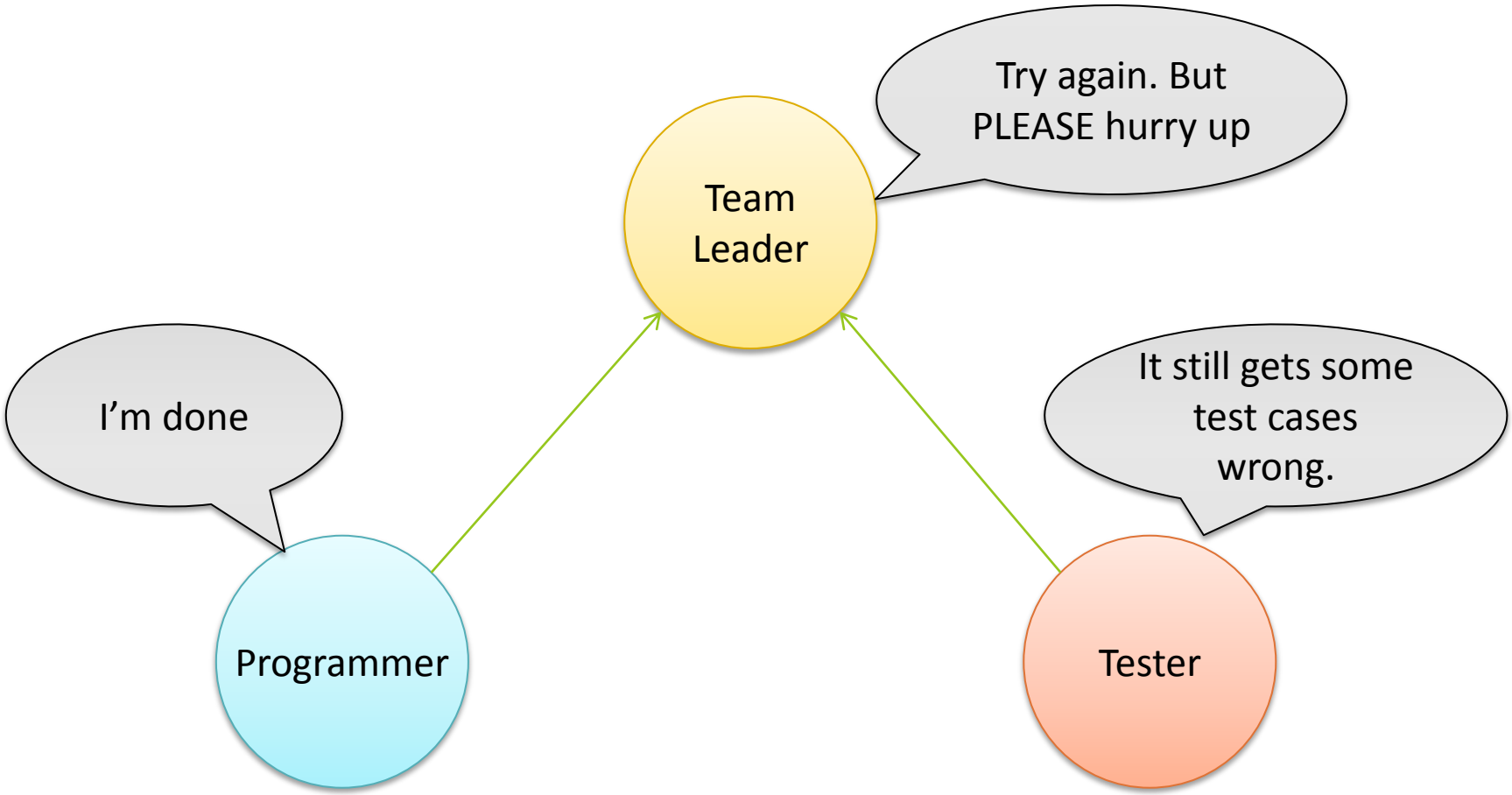
Problem of testing only at the last stage



Problem of testing only at the last stage



Problem of testing only at the last stage



Problem of testing only at the last stage