Informatics 43

LECTURE 13

"HOW DO WE KNOW THE SOFTWARE WORKS? (PART 1)"

Last Time

- We use HCI/UCD methods…
  - Interviews/observations, personas, scenarios, storyboards, mockups, design guidelines, heuristic evaluation, user testing
- …for good reasons
  - Sales increase
  - Performance increases
  - Traffic counts increase
- It's all about the user!

Today's lecture – **How do we know the software works?**

- Failures: a second look
- Quality assurance
- Testing: who, what, how

Today's lecture – **How do we know the software works?**

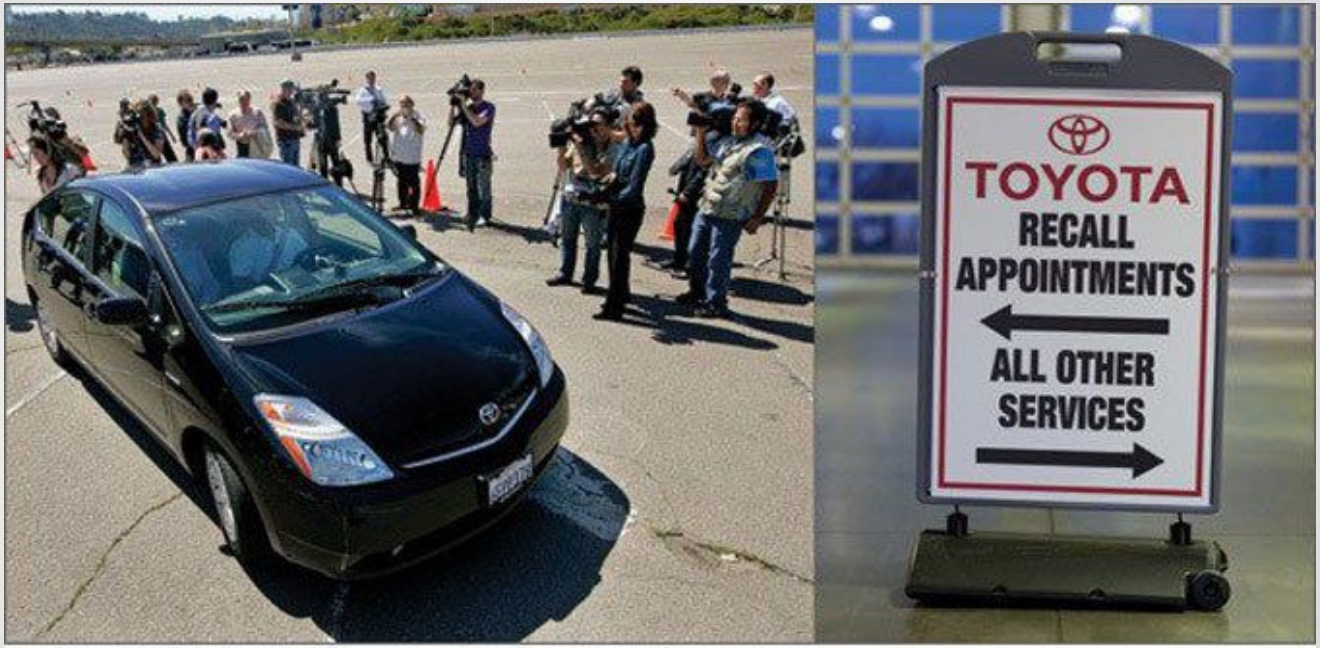
- Failures: a second look
- Quality assurance
- Testing: who, what, how

Boeing 737 Max

"The 737 Max saga teaches us not only about the limits of technology and the risks of complexity, it teaches us about our real priorities. Today, safety doesn't come first—money comes first, and safety's only utility in that regard is in helping to keep the money coming. The problem is getting worse because our devices are increasingly dominated by something that's all too easy to manipulate: software."

Source: https://spectrum.ieee.org/how-the-boeing-737-max-disaster-looks-to-a-software-developer

Toyota "Unintended Acceleration"

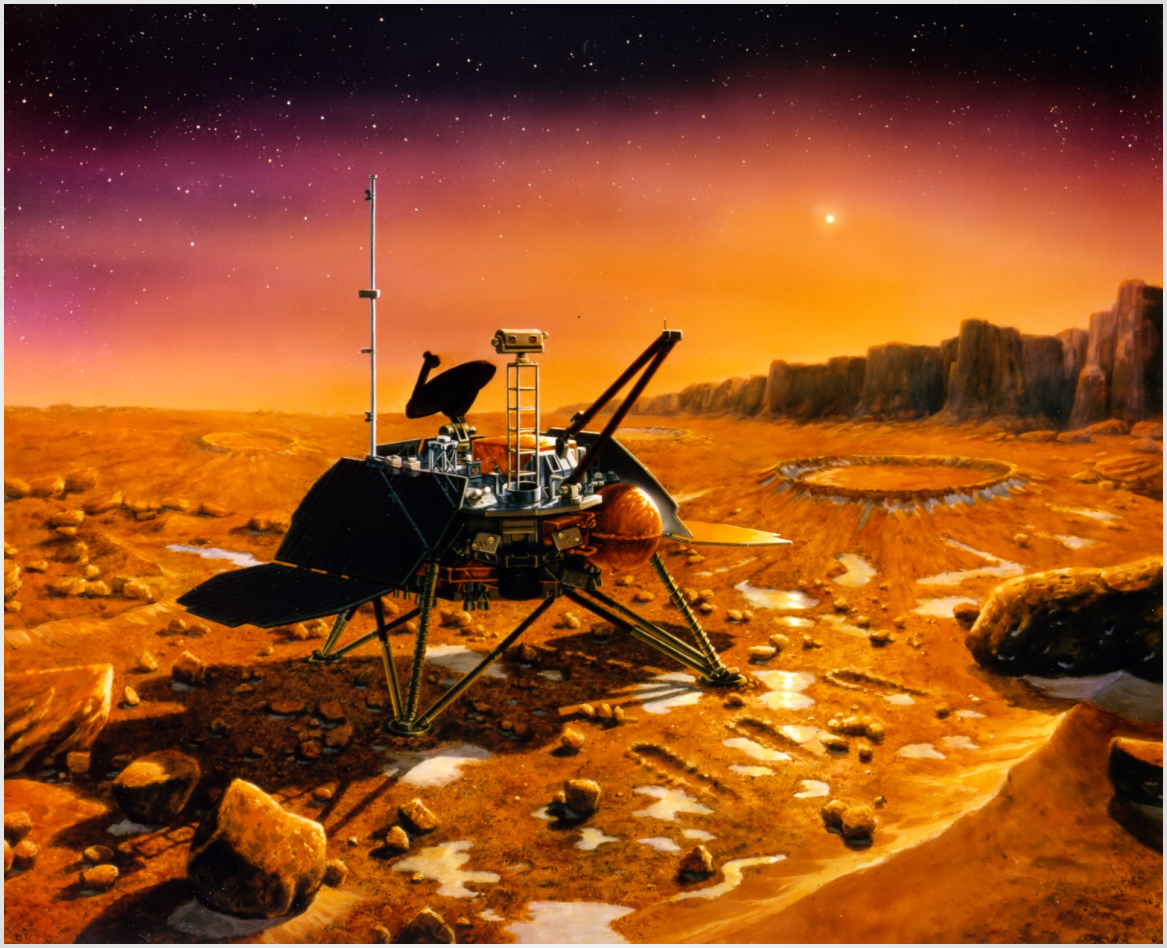


FROM LEFT: MEDIA STAND IN A PARKING LOT, WAITING FOR A PRIUS TO RUN AWAY; RECALLS KEPT TOYOTA SERVICE BAYS PLENTY BUSY

Source: https://www.caranddriver.com/features/its-all-your-fault-the-dot-renders-its-verdict-on-toyotas-unintended-acceleration-scare-feature

Apollo 8

Mars Polar Lander



Y2K

- Bug description
  - Date formats were MM/DD/YY, e.g., 01/01/98, 02/02/99, 03/03/00
  - Does 00 mean 2000 or 1900?
  - Does 1999 turn to 19100?
- Effects
  - Relatively minor
- Cost: $300 billion!

https://codeofmatt.com/list-of-2020-leap-day-bugs/

NASA's "Cardinal Rules for Safety"

- No single event or action shall be allowed to initiate a potentially hazardous event.
- When an unsafe condition or command is detected, the system shall
  - Inhibit the potentially hazardous event sequence.
  - Initiate procedures or functions to bring the system to a predetermined "safe" state.

For the full 388-page doc, see:
http://www.hq.nasa.gov/office/codeq/doctree/871913.pdf

The Power of Ten
10 Rules for Writing Safety Critical Code

1. Restrict to simple control flow constructs.
2. Give all loops a fixed upper-bound.
3. Do not use dynamic memory allocation after initialization.
4. Limit functions to no more than 60 lines of text.
5. Use minimally two assertions per function on average.
6. Declare data objects at the smallest possible level of scope.
7. Check the return value of non-void functions, and check the validity of function parameters.
8. Limit the use of the preprocessor to file inclusion and simple macros.
9. Limit the use of pointers. Use no more than two levels of dereferencing per expression.
10. Compile with all warnings enabled, and use one or more source code analyzers.

Based on: "The Power of Ten -- Rules for Developing Safety Critical Code," *IEEE Computer*, June 2006, pp. 93-95 (PDF).

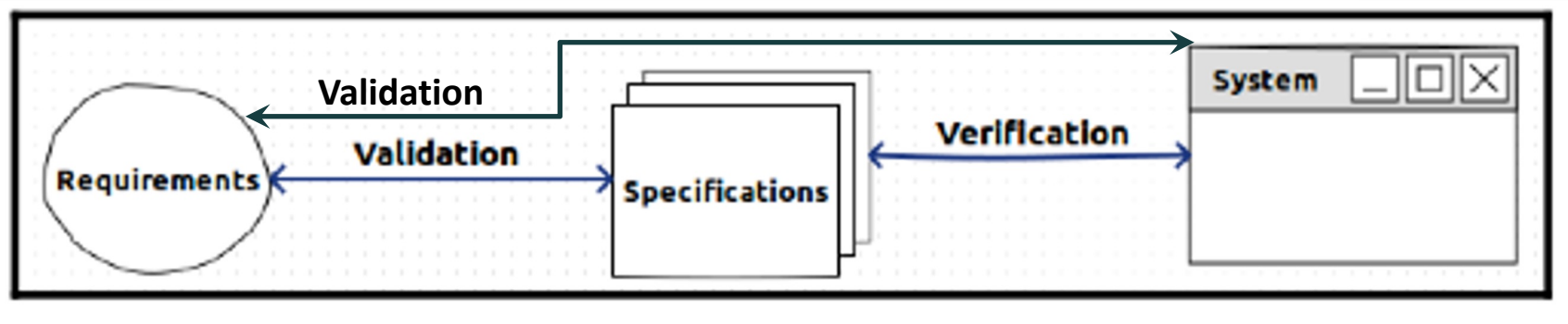
Source: http://spinroot.com/p10/

Today's lecture – **How do we know the software works?**

- Failures: a second look
- Quality assurance
- Testing: who, what, how

QA Goals: Verification and Validation

- Quality Assurance = All activities designed to measure and improve quality in a product



- Verification: Does it conform to specifications?
- Validation: Does it serve its purpose?

QA Techniques

- Formal methods
- Static analysis of program properties
- Reviews and inspections
- Testing

Use a mixture of techniques!

Today's lecture – **How do we know the software works?**

- Failures: a second look
- Quality assurance
- Testing: who, what, how

Testing – Basic Process

- Detect and correct bugs in a software product
- Exercise a module, collection of modules, or system
  - Devise test case (input)
    - Create expected output
  - Run test case
  - Capture actual output
  - Compare actual output to expected output

Actual output = expected output	**Actual output ≠ expected output**
Test case **SUCCEEDS** or **PASSES**	Test case **FAILS** (report failure)

- (Lather, rinse, and repeat)

Much of the testing process is automated

```python
import unittest

class TestStringMethods(unittest.TestCase):

    def test_upper(self):
        self.assertEqual('foo'.upper(), 'FOO')

    def test_isupper(self):
        self.assertTrue('FOO'.isupper())
        self.assertFalse('Foo'.isupper())

    def test_split(self):
        s = 'hello world'
        self.assertEqual(s.split(), ['hello', 'world'])
        # check that s.split fails when the separator is not a string
        with self.assertRaises(TypeError):
            s.split(2)

if __name__ == '__main__':
    unittest.main()
```

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Source: https://docs.python.org/2/library/unittest.html

Testing Goals

- Find and fix failures/faults/errors
- Improve confidence that the system performs as specified (verification) and as desired (validation)
- All in a manner that is
  - Accurate
  - ~~Complete~~ Thorough
  - Repeatable
  - Systematic

Program testing can be used to show the presence of bugs, but never to show their absence [Dijkstra]

Attendance Quiz

Dijkstra Quote

Testing Terminology

From IEEE610.12-90 (IEEE Standard Glossary of Software Engineering Terminology):

- **Mistake**: A human action that produces an incorrect result ("mistake" applies to both coder and user)
- **Fault / Bug**: An incorrect step, process, or data definition in a computer program (something you can point to in the code)
- **Error**: A difference between a computed result and the correct result (could be internal or external)
- **Failure**: The [incorrect] result of a fault (externally visible unexpected behavior or output)

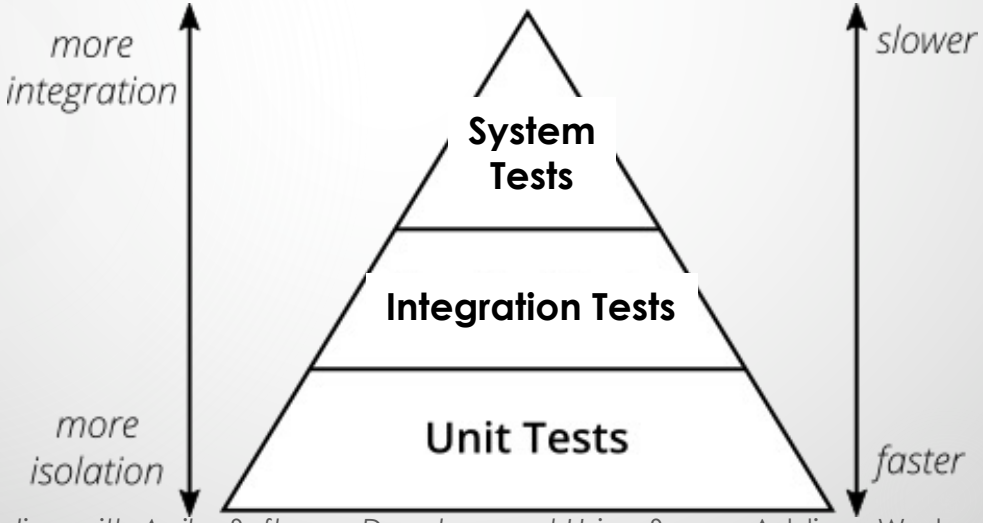
The common term "defect" usually means fault.

Who does the testing?

- Programmers
- Testers
- Users
  - Acceptance testing
  - Alpha testing
  - Beta testing
  - Crowdsourcing/bug bounties

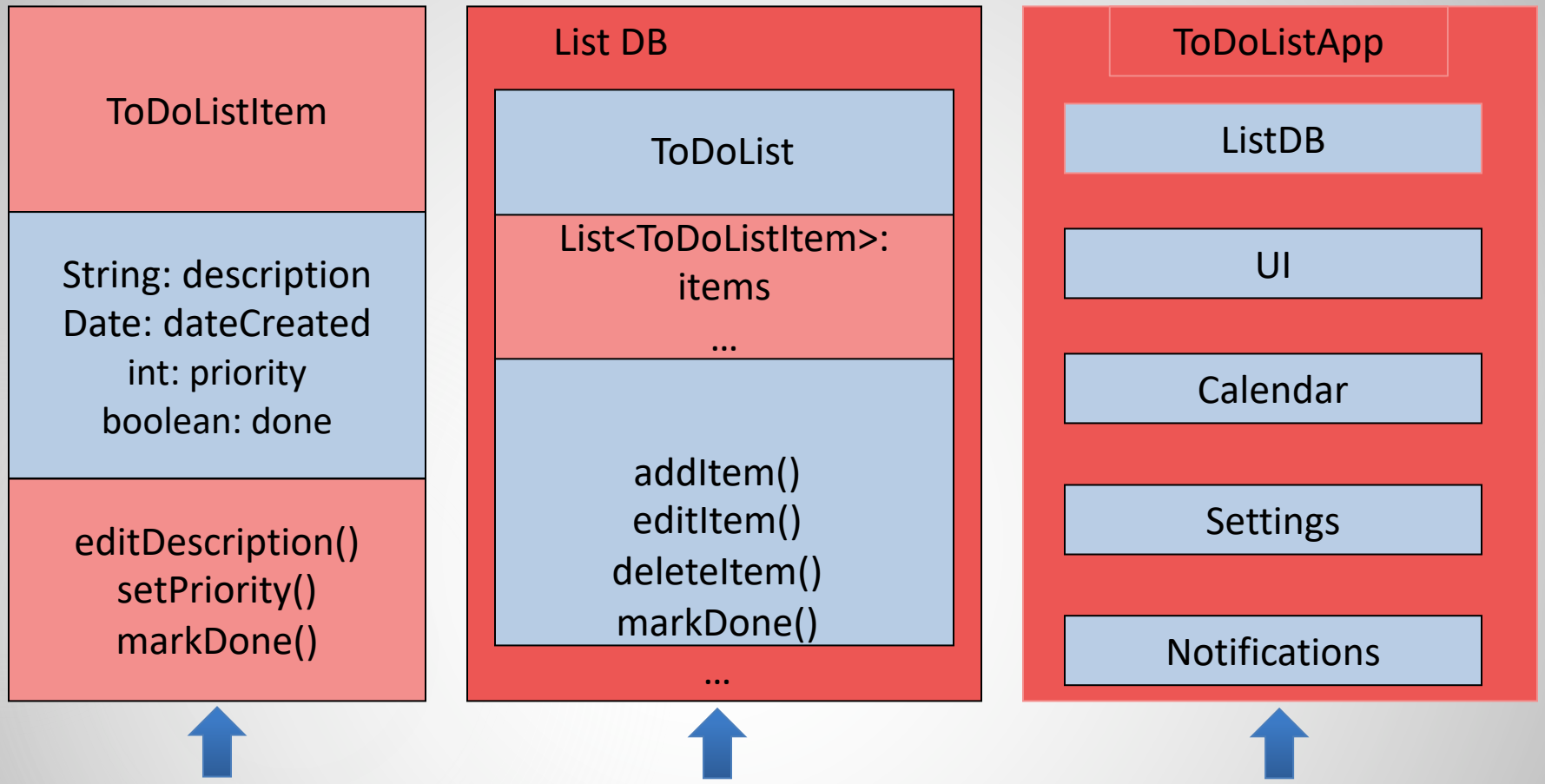
Levels of Testing

- Unit testing
  - Testing of a single code unit
- Functional/integration testing
  - Testing of interfaces among integrated units
- System/acceptance testing
  - Testing of complete system for satisfaction of requirements



Source: M. Cohn, *Succeeding with Agile: Software Development Using Scrum*, Addison-Wesley Professional, 2009.

Levels of Testing



Unit testing

ToDoListItem.setPriority()

Functional/integration testing

ToDoList.editItem()

System/acceptance testing

Create a new list

Edit list info

Edit list item

Delete list

Summary

- Many failures are caused by a lack of good quality assurance (QA) practices
- QA = All activities designed to measure and improve quality in a product
  - Validation & verification
- Testing is the most common QA activity
  - Different levels: unit/functional/system
  - Goal: find and fix failures/faults/errors

Next Time

- Testing, part 2