Introduction to Software Testing
(*2nd edition*)
Chapter 1

Why Do We Test Software?

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

Updated August 2018
First version, 28 August 2011

Testing in the 21st Century

- Software defines behavior
  - network routers, finance, switching networks, other infrastructure
- Today's software market :
  - is much bigger
  - is more competitive
  - has more users

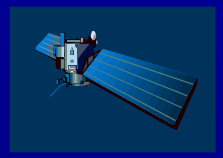
Industry is going through a revolution in what testing means to the success of software products

- Embedded Control Applications
  - airplanes, air traffic control
  - spaceships
  - watches
  - ovens
  - remote controllers
  - PDAs
  - memory seats
  - DVD players
  - garage door openers
  - cell phones
- Agile processes put increased pressure on testers
  - Programmers must unit test – with no training or education!
  - Tests are key to functional requirements

Software is a Skin that Surrounds Our Civilization

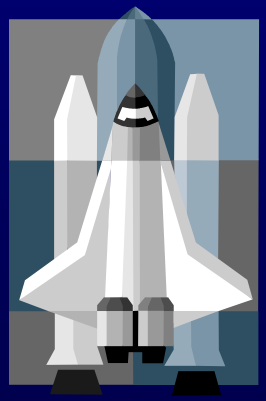
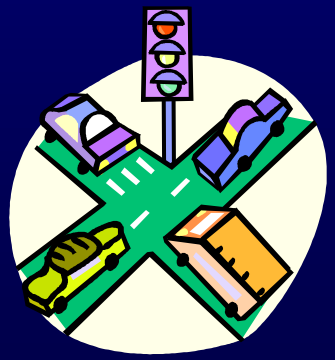










Quote due to Dr. Mark Harman

Software Faults, Errors & Failures

- Software Fault : A static defect in the software
- Software Error : An incorrect internal state that is the manifestation of some fault
- Software Failure : External, incorrect behavior with respect to the requirements or other description of the expected behavior

Faults in software are equivalent to design mistakes in hardware.

Software does not degrade.

Fault and Failure Example

- A patient gives a doctor a list of symptoms
  - Failures
- The doctor tries to diagnose the root cause, the ailment
  - Fault
- The doctor may look for anomalous internal conditions (high blood pressure, irregular heartbeat, bacteria in the blood stream)
  - Errors

Most medical problems result from external attacks (bacteria, viruses) or physical degradation as we age.

Software faults were there at the beginning and do not "appear" when a part wears out.

A Concrete Example

```
public static int numZero (int[] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the number of occurrences of 0 in arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
      if (arr[i] == 0)
         count++;
   return count;
}
```

- There is a simple fault in `numZero`
- Where is the fault location in the source code?
- How would you fix it?

Example – Let's Analyze

```
public static int numZero (int[] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the number of occurrences of 0 in arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
      if (arr[i] == 0)
         count++;
   return count;
}
```

- Fault: a defect in source code

  `i = 1` [should start searching at 0, not 1]

- Error: erroneous program state caused by execution of the defect

  `i` is 1, not 0, on the first iteration [array entry 0 is not ever read]

- Failure: propagation of erroneous state to the program outputs

  Happens as long as `arr.length > 0` and `arr[0] = 0`

Example – Test Cases

```
public static int numZero (int[] arr)
{  // Effects: If arr is null throw NullPointerException
   // else return the number of occurrences of 0 in arr
   int count = 0;
   for (int i = 1; i < arr.length; i++)
      if (arr[i] == 0)
         count++;
   return count;
}
```

Fault: `i = 1` [should start searching at 0, not 1]

- Test 1: [4, 6, 0], expected 1

  Error: `i` is 1, not 0, on the first iteration

  Failure: none

- Test 2: [0, 4, 6], expected 1

  Error: `i` is 1, not 0, error propagates to the variable `count`

  Failure: `count` is 0 at the return statement

Example – State Representation

```
   public static int numZero (int[] arr)
   {  // Effects: If arr is null throw NullPointerException
      // else return the number of occurrences of 0 in arr
L1    int count = 0;
L2    for (int i = 1; i < arr.length; i++)
L3       if (arr[i] == 0)
L4          count++;
L5    return count;
   }
```

- Assume that we want to represent program states using the notation <var_1 = v_1, …, var_n = v_n, PC = program counter>

- Sequence of states in the execution of `numZero({0, 4, 6})`

  1: < arr={0, 4, 6}, PC=[`int count=0` (L1)] >

  2: < arr={0, 4, 6}, `count=0`, PC=[`i=1` (L2)] >

  3: < arr={0, 4, 6}, `count=0`, `i=1`, PC=[`i<arr.length` (L2)] >

  …

  < arr={0, 4, 6}, `count=0`, PC=[`return count;` (L5)] >

Example – Error State

- Error state
  - The first different state in execution in comparison to an execution to the state sequence of what would be the correct program

- If the code had i=0 (correct program), the execution of `numZero({0, 4, 6})` would be

  1: < arr={0, 4, 6}, PC=[`int count=0` (L1)] >
  2: < arr={0, 4, 6}, `count=0`, PC=[`i=0` (L2)] >
  3: < arr={0, 4, 6}, `count=0`, `i=0`, PC=[`i<arr.length` (L2)] >
  …

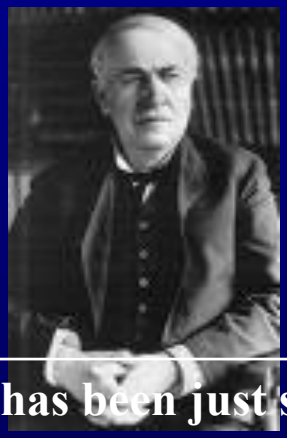
- Instead, we have

  1: < arr={0, 4, 6}, PC=[`int count=0` (L1)] >
  2: < arr={0, 4, 6}, `count=0`, PC=[`i=1` (L2)] >
  3: < arr={0, 4, 6}, `count=0`, `i=1`, PC=[`i<arr.length` (L2)] >
  …

The first error state is immediately after `i=1` in line L2

The Term Bug

- *Bug* is used informally
- Sometimes speakers mean fault, sometimes error, sometimes failure … often the speaker doesn't know what it means !
- This class will try to use words that have precise, defined, and unambiguous meanings

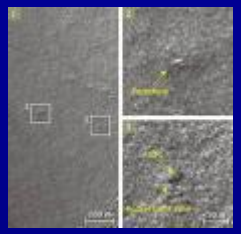


"It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and *[it is]* then that 'Bugs'—as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite. . ." – Thomas Edison

"an analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders. " – Ada, Countess Lovelace (notes on Babbage's Analytical Engine)

Spectacular Software Failures

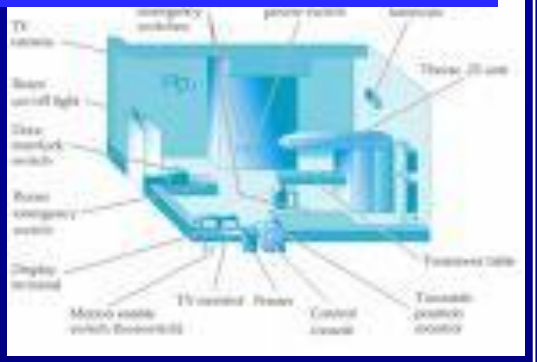
- NASA's Mars lander: September 1999, crashed due to a units integration fault



Mars Polar Lander crash site?

- THERAC-25 radiation machine : Poor testing of safety-critical software can cost *lives* : 3 patients were killed
- Ariane 5 explosion : Millions of $$
- Intel's Pentium FDIV fault : Public relations nightmare

THERAC-25 design



Ariane 5: exception-handling fault : forced self destruct on maiden flight (64-bit to 16-bit conversion: about 370 million $ lost)

We need our software to be *dependable*
Testing is *one* way to assess dependability

Northeast Blackout of 2003

508 generating units and 256 power plants shut down

Affected 10 million people in Ontario, Canada

Affected 40 million people in 8 US states

Financial losses of $6 Billion USD

The alarm system in the energy management system failed due to a software error and operators were not informed of the power overload in the system

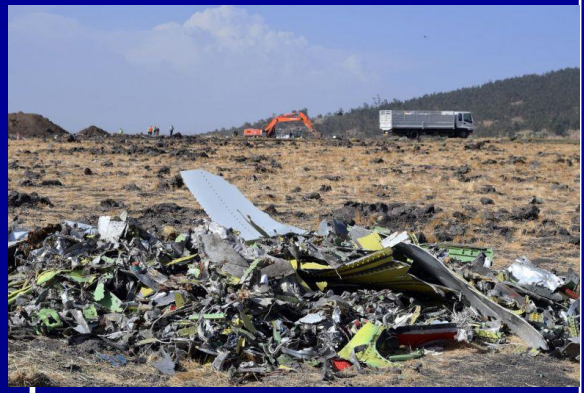
Costly Software Failures

- NIST report, "The Economic Impacts of Inadequate Infrastructure for Software Testing" (2002)
  - Inadequate software testing costs the US alone between $22 and $59 billion annually
  - Better approaches could cut this amount in half
- Huge losses due to web application failures
  - Financial services : $6.5 million per hour (just in USA!)
  - Credit card sales applications : $2.4 million per hour (in USA)
- In Dec 2006, *amazon.com's* BOGO offer turned into a double discount
- 2007 : Symantec says that most security vulnerabilities are due to faulty software

World-wide monetary loss due to poor software is staggering

Spectacular software Failures

- Boeing A220 : Engines failed after software update allowed excessive vibrations



- Boeing 737 Max : Crashed due to overly aggressive software flight overrides (MCAS)
- Toyota brakes : Dozens dead, thousands of crashes

- Healthcare website : Crashed repeatedly on launch—never load tested
- Northeast blackout : 50 million people, $6 billion USD lost … alarm system failed

Software testers try to find faults before the faults find users

The True Cost of Software Failure

Fail watch analyzed news articles for 2016

- 606 reported software failures
- Impacted half the world's population
- Cost a combined $1.7 trillion US dollars

Poor software is a significant drag on the world's economy

Not to mention frustrating

What Does This Mean?

Software testing is getting more important

What are we trying to do when we test ?

What are our goals ?

Validation & Verification (*IEEE*)

- Validation : The process of evaluating software at the end of software development to ensure compliance with intended usage

- Verification : The process of determining whether the products of a given phase of the software development process fulfill the requirements established during the previous phase

IV&V stands for "*independent verification and validation*"

Tactical Goals : Why Each Test ?

If you don't know why you're conducting each test, it won't be very helpful

- Written test objectives and requirements must be documented
- What are your planned coverage levels?
- How much testing is enough?
- Common objective – spend the budget . . . test until the ship-date . . .
  - Sometimes called the "date criterion"

Why Each Test ?

If you don't start planning for each test when the functional requirements are formed, you'll never know why you're conducting the test

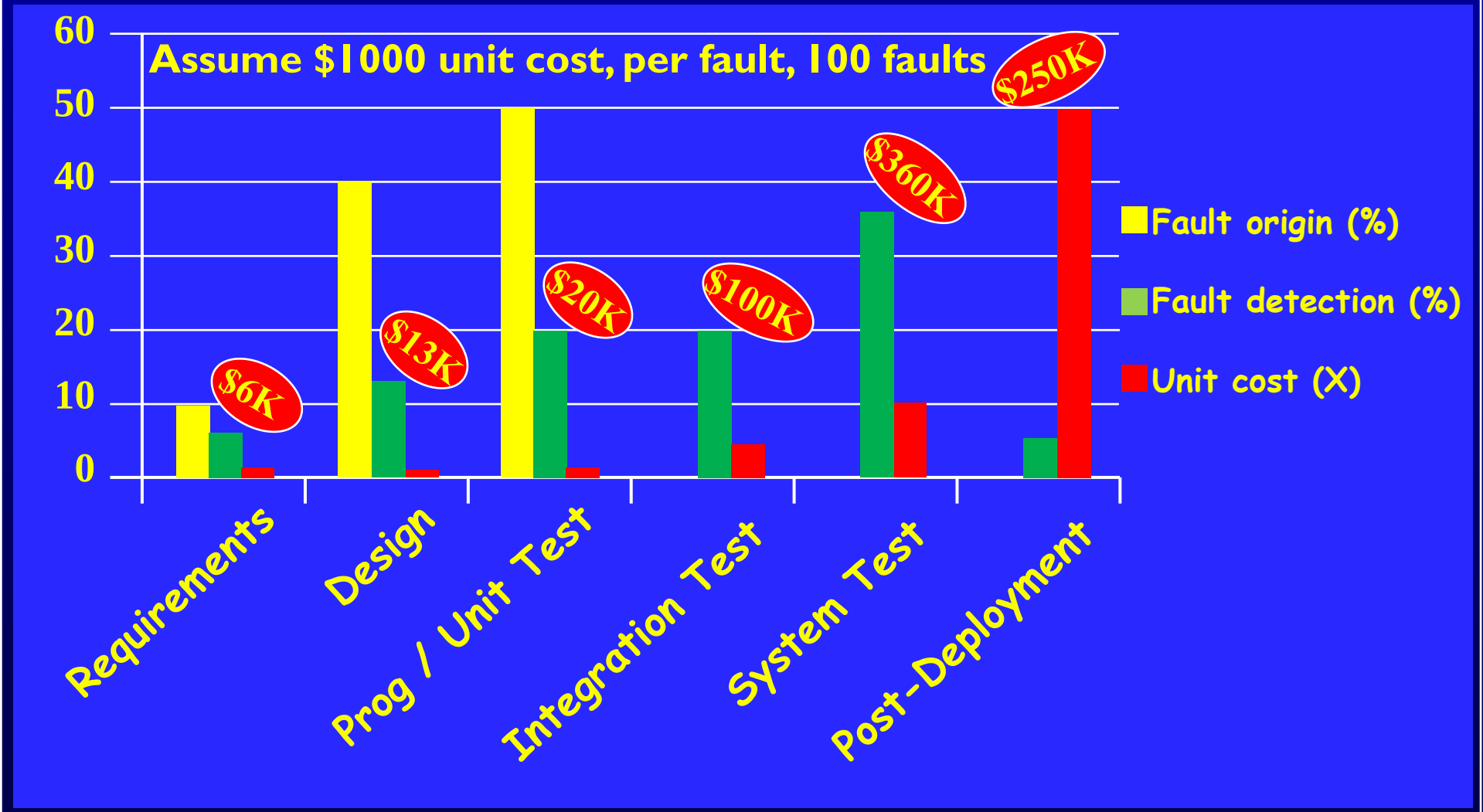
- What fact does each test try to verify?
- Requirements definition teams need testers!

Cost of Not Testing

Poor Program Managers might say:
"Testing is too expensive."

- Testing is the most time consuming and expensive part of software development
- Not testing is even more expensive
- If we have too little testing effort early, the cost of testing increases
- Planning for testing after development is prohibitively expensive

Cost of Late Testing



Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002

Summary: Why Do We Test Software ?

A tester's goal is to eliminate faults as early as possible

- **Improve quality**
- **Reduce cost**
- **Preserve customer satisfaction**