Testing
Object-Oriented Systems

What is Testing?

Marcello Bonsangue
Frank de Boer
Software may contain errors (1843)

- “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 Byron, Countess of Lovelace (notes on Babbage’s Analytical Engine)
The first bug

“...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, 1878
The first actual bug is found

- A moth trapped in a relay. Grace Hopper, 1947
Why we do tests?

- To avoid failures in software
  - NASA Mars Polar Lander crash due to a software integration fault (misunderstanding in unit measures)
  - Arianne 5 explosion due to a fault exception-handling
  - Therac-25 radiation machine: people died due to a fault interface

- We want programs to be reliable!!
Do You Trust Your System?

*The real wonder is that the system works as well as it does*

(Peterson, 1996)

but remember that software systems provide the infrastructure in virtually all industries today:

- air traffic control
- Automotive systems
- water level management
- energy production and distribution
- …
How do We Build Trust in Our Systems?

- Any **engineering process** consists of
  - construction activities
  - techniques **to check** intermediate and final products

- **Testing** is one technique to increase our **confidence** in the **correctness** of a software system.
Validation and Verification

Validation: The process of evaluating software at the end of software development to ensure compliance with intended usage
- Are we building the right system?

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
- Are we building the system right?
Software verification takes various forms

- Model checking
- Deductive verification
- Testing
- Static analysis
- …
Software verification

- Software has some characteristics that make verification difficult:
  - Many different quality requirements
  - Evolving structure
  - Uneven distribution of faults

- If an **elevator** can safely carry a load of 1000 kg, it can also safely carry any smaller load

- If a **procedure** correctly sorts a set of 256 elements, it may fail on a set of 255 or 53 or 12 elements, as well as on 257 or 1023
What is Testing?

- **Testing** is the process of identifying **defects**

- **A defect** is any variance between actual and expected observable results.
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.
Software Faults, Errors & Failures

Patient describes symptoms (failures)

Doctor must find cause of symptoms (fault)

It looks for anomalous conditions, eg. high blood pressure, or high cholesterol (errors)
An example

Public static int numZero (int[ ] x) {
// return the number of occurrences of 0 in array x
int count = 0
for (int i=1; i < x.length; i++) {
    if (x[i] == 0) {
        count ++
    }
}
return count
}

Test x = [2,7,0]
Expected count =1

Test x = [0,2,7]
Expected count =1
An example

Public static int numZero (int[ ] x) {
   // return the number of occurrences of 0 in array x
   int count = 0;  
   for (int i=1; i < x.length; i++) {
      if (x[i] == 0) {
         count ++
      }
   }
   return count
}

Both executions result in an error (the fault is executed) but only the second is a failure.
Test Cases

- **Test case values**: the input necessary to complete an execution of a program

- **Expected results**: The result that will be produced when executing the test if the program satisfies its intended behavior

- **Test case**: The test case values, expected results, any other inputs necessary to start and conclude the execution

- Test cases are used to determine if a program satisfies a test requirement
Test Suite

- **Test suite**: a set of test cases

- **Executable test script**: A test case that is prepared in a form to be executed automatically on the test software and produce a report
Constructing a Test Suite

- **Test coverage**: define a model of the software, then find ways to cover it

- **Test requirements**: specific things that must be satisfied or covered during testing
  - Requirements that cannot be satisfied are called **infeasible**

- **Coverage Criterion**: A collection of rules and a process that define test requirements on a test suite
An example: Jelly beans

- Lemon, Apricot: Yellow
- Orange, Tangerine: Orange
- Pistachio: Green
- Pear: White

Test requirement
- All flavors: \{Lemon, Orange, Tangerine, Apricot, Pistachio, Pear\}
- All colors: \{Yellow, Orange, Green, White\}
- Infeasible test requirement: \{Orange, Purple\}

Coverage Criterion: A strategy for selecting jelly beans
Adequacy of a test suite

- Ideally – exhaustively test everything
- Practically impossible

- Test coverage criteria are measures of adequacy to increase the confidence that we have tested enough
Coverage

- Given a set of test requirements \( TR \) for coverage criterion \( C \), a test suite \( T \) satisfies \( C \) coverage if and only if for every test requirement \( tr \) in \( TR \), there is at least one test \( t \) in \( T \) such that \( t \) satisfies \( tr \).

- A coverage criterion \( C_1 \) subsumes \( C_2 \) if and only if every test set that satisfies \( C_1 \) also satisfies \( C_2 \).