Fault, Error, and Failure

Testing, Quality Assurance, and Maintenance
Winter 2019

Prof. Arie Gurfinkel

based on slides by Prof. Lin Tan and others
Terminology, IEEE 610.12-1990

**Fault** -- often referred to as **Bug** [Avizienis’00]
- A static defect in software (incorrect lines of code)

**Error**
- An incorrect internal state (unobserved)

**Failure**
- External, incorrect behaviour with respect to the expected behaviour (observed)

Not used consistently in literature!
What is this?

A fault?

An error?

A failure?

We need to describe specified and desired behaviour first!
Erroneous State ("Error")
Design Fault
Mechanical Fault
Example: Fault, Error, Failure

```java
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;
```

```
<table>
<thead>
<tr>
<th>State of the program:</th>
<th>x, i, count, PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error State:</td>
<td></td>
</tr>
<tr>
<td>x = [2,7,0]</td>
<td></td>
</tr>
<tr>
<td>i =1</td>
<td></td>
</tr>
<tr>
<td>count =0</td>
<td></td>
</tr>
<tr>
<td>PC=first iteration for</td>
<td></td>
</tr>
<tr>
<td>Expected State:</td>
<td></td>
</tr>
<tr>
<td>x = [2,7,0]</td>
<td></td>
</tr>
<tr>
<td>i =0</td>
<td></td>
</tr>
<tr>
<td>count =0</td>
<td></td>
</tr>
<tr>
<td>PC=first iteration for</td>
<td></td>
</tr>
</tbody>
</table>
```

Fix: ```for(int i=0; i<x.length; i++)```  
```
x = [2,7,0], fault executed, error, no failure
x = [0,7,2], fault executed, error, failure
```

State of the program: x, i, count, PC
Exercise: The Program

/* Effect: if x==null throw NullPointerException. Otherwise, return the index of the last element in the array ‘x’ that equals integer ‘y’. Return -1 if no such element exists. */

public int findLast (int[] x, int y) {
    for (int i=x.length-1; i>0; i--) {
        if (x[i] == y) { return i; }
    }
    return -1;
}

/* test 1: x=[2,3,5], y=2;
   expect: findLast(x,y) == 0
 test 2: x=[2,3,5,2], y=2;
   expect: findLast(x,y) == 3 */
Exercise: The Problem

Read this faulty program, which includes a test case that results in failure. Answer the following questions.

• (a) Identify the fault, and fix the fault.
• (b) If possible, identify a test case that does not execute the fault.
• (c) If possible, identify a test case that executes the fault, but does not result in an error state.
• (d) If possible identify a test case that results in an error, but not a failure. Hint: Don't forget about the program counter.
• (e) For the given test case ‘test1’, identify the first error state. Be sure to describe the complete state.
States

State 0:
- $x = [2,3,5]$
- $y = 2$
- $i = \text{undefined}$
- PC = findLast(...)

State 1:
- $x = [2,3,5]$
- $y = 2$
- $i = \text{undefined}$
- PC = before $i = x.\text{length}-1$

State 2:
- $x = [2,3,5]$
- $y = 2$
- $i = 2$
- PC = after $i = x.\text{length}-1$

State 3:
- $x = [2,3,5]$
- $y = 2$
- $i = 2$
- PC = $i > 0$
States

- **State 3:**
  - \( x = [2,3,5] \)
  - \( y = 2 \)
  - \( i = 2 \)
  - \( PC = i>0; \)

- **State 4:**
  - \( x = [2,3,5] \)
  - \( y = 2 \)
  - \( i = 2 \)
  - \( PC = \text{if} \ (x[i] == y); \)

- **State 5:**
  - \( x = [2,3,5] \)
  - \( y = 2 \)
  - \( i = 1 \)
  - \( PC = i--; \)

- **State 6:**
  - \( x = [2,3,5] \)
  - \( y = 2 \)
  - \( i = 1 \)
  - \( PC = i>0; \)

- **State 7:**
  - \( x = [2,3,5] \)
  - \( y = 2 \)
  - \( i = 1 \)
  - \( PC = \text{if} \ (x[i] == y); \)

- **State 8:**
  - \( x = [2,3,5] \)
  - \( y = 2 \)
  - \( i = 0 \)
  - \( PC = i--; \)
States

Incorrect Program

• State 8:
  • x = [2,3,5]
  • y = 2
  • i = 0
  • PC = i--; 

• State 9:
  • x = [2,3,5]
  • y = 2
  • i = 0
  • PC = i>0;

• State 10:
  • x = [2,3,5]
  • y = 2
  • i = 0
  • PC = return -1;

Correct Program

• State 10:
  • x = [2,3,5]
  • y = 2
  • i = 0
  • PC = if (x[i]==y);
Exercise: Solutions (1/2)

(a) The for-loop should include the 0 index:
   • for (int i=x.length-1; i >= 0; i--)

(b) The null value for x will result in a NullPointerException before the loop test is evaluated, hence no execution of the fault.
   • Input: x = null; y = 3
   • Expected Output: NullPointerException
   • Actual Output: NullPointerException

(c) For any input where y appears in a position that is not position 0, there is no error. Also, if x is empty, there is no error.
   • Input: x = [2, 3, 5]; y = 3;
   • Expected Output: 1
   • Actual Output: 1
Exercise: Solutions (2/2)

(d) For an input where y is not in x, the missing path (i.e. an incorrect PC on the final loop that is not taken, normally \(i = 2, 1, 0\), but this one has only \(i = 2, 1\), ) is an error, but there is no failure.

- Input: \(x = [2, 3, 5]; y = 7\);
- Expected Output: -1
- Actual Output: -1

(e) Note that the key aspect of the error state is that the PC is outside the loop (following the false evaluation of the 0>0 test. In a correct program, the PC should be at the if-test, with index \(i==0\).

- Input: \(x = [2, 3, 5]; y = 2\);
- Expected Output: 0
- Actual Output: -1
- First Error State:
  - \(x = [2, 3, 5]\)
  - \(y = 2\);
  - \(i = 0\) (or undefined);
  - PC = return -1;
RIP Model

Three conditions must be present for an error to be observed (i.e., failure to happen):

- **Reachability**: the location or locations in the program that contain the fault must be reached.
- **Infection**: After executing the location, the state of the program must be incorrect.
- **Propagation**: The infected state must propagate to cause some output of the program to be incorrect.
HOW DO WE DEAL WITH FAULTS, ERRORS, AND FAILURES?
Addressing Faults at Different Stages

- **Fault Avoidance**
  - Better Design, Better PL, ...

- **Fault Detection**
  - Testing, Debugging, ...

- **Fault Tolerance**
  - Redundancy, Isolation, ...
Declaring the Bug as a Feature
Modular Redundancy: Fault Tolerance
Patching: Fixing the Fault
Testing: Fault Detection
Testing vs. Debugging

Testing: Evaluating software by observing its execution

Debugging: The process of finding a fault given a failure

Testing is hard:
• Often, only specific inputs will trigger the fault into creating a failure.

Debugging is hard:
• Given a failure, it is often difficult to know the fault.
Testing is hard

```c
if ( x - 100 <= 0 )
    if ( y - 100 <= 0 )
        if ( x + y - 200 == 0 )
            crash();
```

Only input $x=100$ & $y=100$ triggers the crash.

If $x$ and $y$ are 32-bit integers, what is the probability of a crash?

- $1 / 2^{64}$
Exercise: The Problem

```python
def pos_odd(x):
    """Ensures: returns the number of positive odd elements in the list x
    or throws an exception if x is not a list of numbers""
    cnt = 0
    i = 0
    while i < len(x):
        if x[i] % 2 == 1:
            cnt = cnt + 1
        i = i + 1
    return cnt
```

1. What is the fault in this program
2. Identify a test case that does not execute the fault
3. Identify a test case that results in an error but does not cause failure
4. Identify a test case that causes a failure but no error
5. For the test case \( x = [-10, -9, 0, 99, 100] \) the expected output is 1. Identify the first error state
Exercise: Solution

a) Fault is at line 7. Negative numbers are not considered. Fixed by
   \[
   \text{if } x[i] > 0 \text{ and } x[i] \% 2 == 1
   \]

b) Any input that does not execute line 7. For example, \( x = 7 \) (not a list of numbers), \( x=[] \) (empty list), etc.

c) Any list that contains numbers and not-numbers. At a non-number, an exception is thrown (which is expected and is not a failure) even though an error has occurred before. For example, \( x = [-1, \text{'hey'}] \)

d) This situation is impossible. Fault is required for error, error is required for failure. It is possible to have fault without an error, and error without a failure, but not the other way around

e) The first error state is:
   \[
   x = [-10, -9, 0, 99, 100] \quad i = 1
   \]
   \[
   \text{cnt} = 0 \quad \text{pc} = \text{at line 8}
   \]