Software Testing and Debugging

Program Failures and Bugs

Shin Hong

Program and Execution

- A *program* is a composition of instructions and data that defines a function from input values to output values
  - a program can be represented as *code* in different abstraction levels
  - a program does not change once it is constructed (i.e., static)
- An *execution* of a program is a sequence of the instructions executed with an input value
  - an execution is also defined as a sequence of *program states* transitioned by an instruction execution with a part of an input
  - a program state is a mapping from memory locations to values

$$s_0 \xrightarrow{i_0(d_0)} s_1 \xrightarrow{i_{s_1}(d_1)} s_2 \quad \cdots \quad s_j \xrightarrow{i_{s_j}(d_j)} s_{j+1} \quad \cdots \quad \xrightarrow{i_{s_{n-1}}(d_{n-1})} s_n$$

Example

```
l1   i = 2
l2   print "Enter a number"
l3   read x
l4   if i >= x goto l9
l5   y = x % i
l6   if y == 0 goto l11
l7   i = i + 1
l8   goto l4
l9   print "Prime number"
l10  halt
l11  print "Not a prime number"
l12  halt
```

$P = \{l_1, l_2, l_3, \cdots, l_{12}\}$

$s_0 = [i: ?, x: ?, y: ?, pc=l_1]$

l_1: `i = 2`

$s_1 = [i: 2, x: ?, y: ?, pc= l_2]$

l_2: `print "Enter.."`

$s_2 = [i: 2, x: ?, y: ?, pc= l_3]$

l_3: `read x (3)`

$s_3 = [i: 2, x: 3, y: ?, pc= l_4]$

l_4: `if i>=x goto` l_9

$s_4 = [i: 2, x: 3, y: ?, pc= l_5]$

l_5: `y =x % i`

$s_5 = [i: 2, x: 3, y: 1, pc= l_6]$

l_6: `if y==0 goto` l_{11}

$s_6 = [i: 2, x: 3, y: 1, pc= l_7]$

l_7: `i=i+1`

$s_7 = [i: 3, x: 3, y: 1, pc= l_8]$

l_8: `goto` l_4

$s_8 = [i: 3, x: 3, y: 1, pc= l_4]$

$\vdots$

When Do You Say “This Program Does Not Work”

A program *fails* when its execution does not provide what the user requires (or expects) by using a given computer system

What Program Is Required to Do

- A program is required to finish its execution and produce the sound output for every input in a reasonable time
  - functional requirement
    - correctness
    - reliability
    - robustness
  - performance requirement
    - time, memory, energy performance
  - system constraint
    - instruction-level, program language-level, system-level
    - safety, security

Program Failure

- A *program failure* is an execution that violates one of the posed requirements and produces an unexpected result
  - often called as *error*

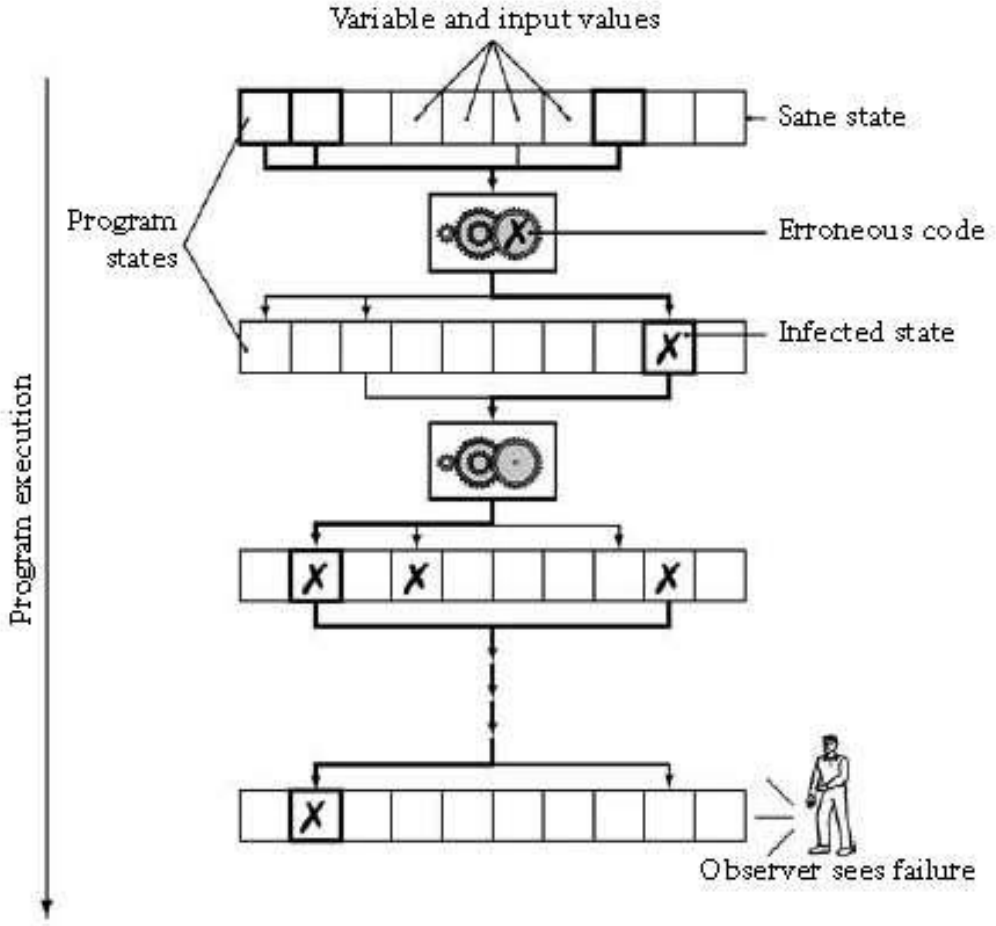
- Different kinds of program failures
  - logic error (asseretion failure)
  - regression failure
  - leak (memory leak, resource leak)
  - memory error (buffer overrun, array out-of-bound, use-after-free)
  - concurrency error
  - hang (indefinite wait, deadlock, timeout)
  - undefined behaviors
  - uncaught exception
  - runtime error, e.g., type error
  - crash (fatal error), e.g., segmentation fault, stack overflow

Program Fault

- A *fault* is a flaw in the program code, that is responsible for making an execution fail
  - often called as a *bug* or *defect*
  - a fault is defined by a part of the program code
- Different kinds of faults
  - regression fault
  - memory fault, concurrency fault, performance fault
  - security vulnerability (weakness)
  - omission fault, multi-hunk fault
  - heisenbug
- c.f. similar concepts: issues, smell, antipatterns

From Fault to Failure

- Execution-Infection-Propagation (PIE): three necessary condition for a fault to result a failure
  - **execution**: an execution reaches to a fault
  - **infection**: the fault induces an invalid state
  - **propagation**: the invalid state leads the execution to produce an observable symptom



Example

```
   mid(x, y, z){
01  m = y;
02  if (m < z)
03    if (m < x)
04      m = x;
05    if (z < m)
06      m = z;
07  else
08    if(x > y)
09      m = z; // m = y;
10    else if (x > z)
11      m = x;
12  print m ;
   }
```

mid(3,3,5)	**mid(4,3,3)**	**mid(4,3,2)**
01 m = 3	01 m = 3	m = 3
02 (m < 5)	02 !(m < 3)	!(m < 3)
03 !(m < 3)		
	07 else	07 else
	08(4 > 3)	08 (4 > 3)
	09 **m = 3**	**09 m = 2**
12 print 3	12 **print 3**	**12 print 2**

Another Example

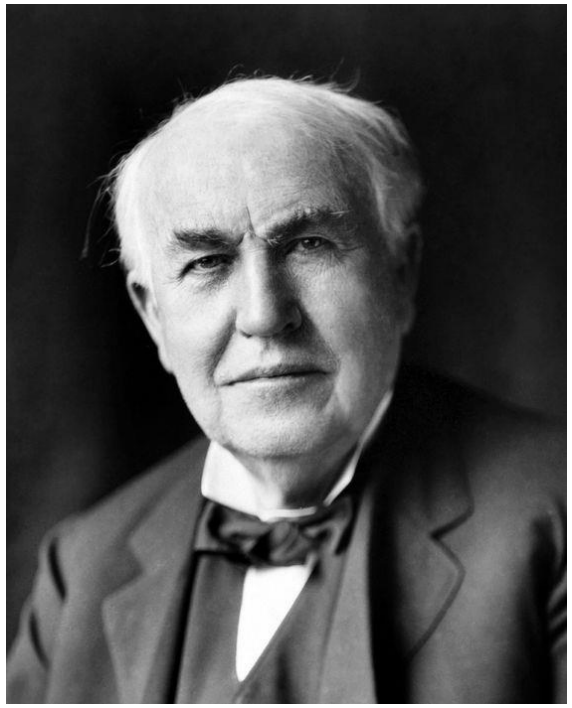

```
int
s_cmp (char *a1, char * a2)
{
  char s1[4] ;
  char s2[4] ;

  for (i = 0 ; a1[i] != 0x0 ; i++)
    s1[i] = a1[i] ;
  s1[i] = 0x0 ;

 for (i = 0 ; a2[i] != 0x0 ; i++)
    s2[i] = a2[i] ;
  s2[i] = 0x0 ;

  return strcmp(s1, s2) ;
}
```

```
s_cmp(“aa”, “aa”)

s_cmp(“aa”, “bb”)

s_cmp(“aaaaa”, “bbbbbb”)

s_cmp(“aaaaa”, “aaaaa”)
```

Debug *verb*

to look for and remove the faults in a computer program

- OALD



Thomas Edison (1847—1931)

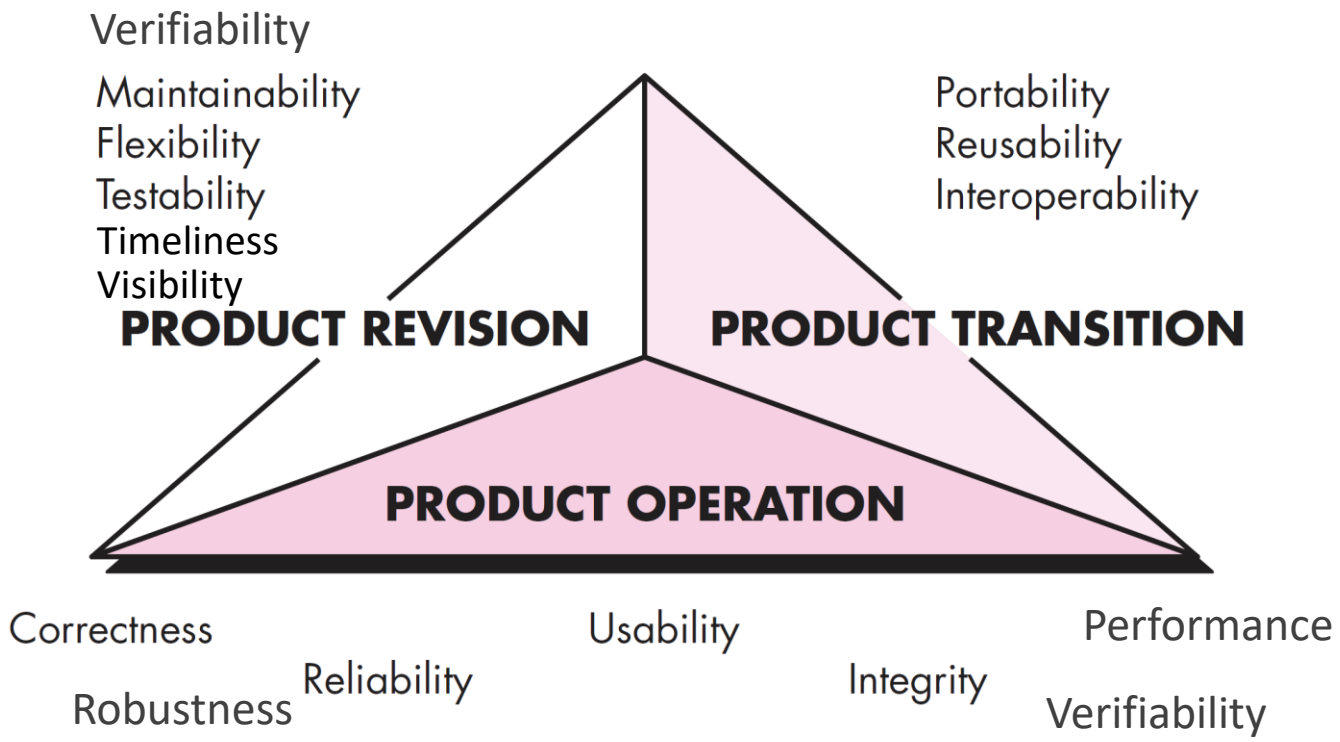
"I did find a '**bug**' […] The inspect appears to find conditions for its existence in all call apparatus of Telephones."

- **not very noticeable**
- **not obvious**
- **possible to kill**

"'**Bug**' […] show themselves, and **months of anxious watching, study, and labor** are requisite before commercial **success—or failure—is certainly reached**."

- **critical**
- **complicated**
- **not manageable**

Software Quality Attributes



McCall's software quality factors (modified)