Testing and Debugging

Aslam Safla
<aslam@cs.uct.ac.za>

Program

Write a program that reads in a list of **test scores** and **classifies** them into ranks -

1, 2+, 2-, 3, FS, F

- and, for each rank, displays:
  - the number of students in the rank (output as a **histogram**).

markClassificationSkeleton.py

```
def main():
    print('='*10) #Prints line to indicate start of program
    print("Enter a list of test scores.")
    i,Fi,uS,S,Th,fS,Fa=0,0,0,0,0,0,0
    firsts=0
    val = input("Enter next test score #"+str(i)+" (press 'Enter' key to stop):")
    while val: #sentinal loop to read in unspecified no. of test scores
        i+=1
        # TODO: add in code to count number in each category
        val =input("Enter next test score #"+str(i)+" (press 'Enter' key to stop):")
    print(" 1:{} 2+:{} 2:{} 3:{} FS:{} F:{} ".format(Fi,uS,S,Th,fS,Fa)) #use of format string
    barChart(Fi,uS,S,Th,fS,Fa) #will print barChart
    print('='*10) #Prints line to indicate end of program
```

UNIVERSITY OF CAPE TOWN

markClassificationSkeleton.py

```
def barChart(Fi,uS,S,Th,fS,Fa):
    total= Fi+uS+S+Th+fS+Fa
    row="{:>2} | {:<10}"#String format for a row in the bar chart
    c='*' #The character to use to print the bars
    barMax=10 #max length of a bar
    #doesn't do anything useful yet


if __name__ == '__main__':  #will explain this again, in more detail
    main()
```

How do we demonstrate that this program is correct?

Errors and testing: Quick Poll

In a typical hour spent programming, how many minutes do you spend fixing errors?

Errors

What is an error?

- When your program does not behave as intended or expected.

What is a bug?

- "…there is a bug in my program …"

Debugging

- the art of removing bugs

Errors and testing: Quick Poll

In a typical hour spent programming, how many minutes do you spend fixing errors?

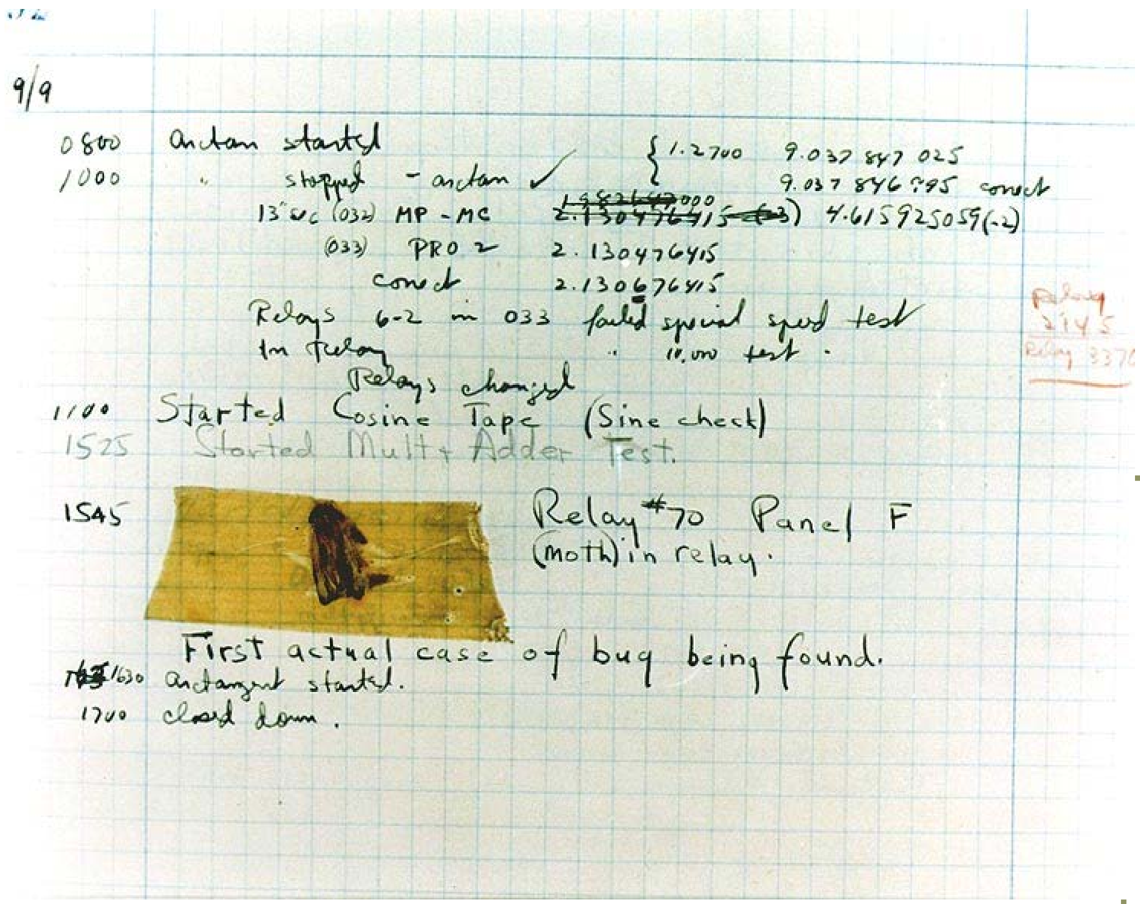
Bugs are unavoidable, **even for the best programmers**

Aim to program so that debugging time is reduced

- for yourself
- and others in the future

This is a matter of both style and strategy

The First "Bug"?



9/9

0800 Anctan started

1000 " stopped - anctan ✓ {1.2700 9.037 847 025
{ 9.037 846 995 conct

13" UC (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (-3) 4.615925059(-2)

(033) PRO 2 2.130476415

conct 2.130676415

Relays 6-2 in 033 failed special speed test

In relay " 10,000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Mult+ Adder Test.

1545 Relay #70 Panel F (moth) in relay.

First actual case of bug being found.

~~1630~~ 1630 antangent started.

1700 closed down.

Relay 2145
Relay 3370

The tale is that the original 'bug' was a moth, which caused a hardware fault in the Harvard Mark I.

The moth was found by Grace Hopper

- Rear Admiral Grace Murray Hopper (December 9, 1906 - January 1, 1992) was an American computer scientist and naval officer
- the first USA computer science "Man of the Year" in 1969.

Reference: http://en.wikipedia.org/wiki/Grace_Hopper

Types of Errors – When (1)

"Compile"-time Error

- Discovered when program is checked, before it is run.
- A result of improper use of Python language
  - usually Syntax Errors.

  e.g. `product = x y`

Types of Errors – When (2)

Run-time Error

- Program structure is correct, but does not execute as expected.

  e.g.

  ```
  x = 0
  y = 15/x
  ```

Examples of Python runtime errors:

- division by zero
- performing an operation on incompatible types
- using an identifier which has not been defined

Types of Errors – Why (1)

- Syntax Error
  - Program does not pass checking/compiling stage.
  - Improper use of Python language.
    - e.g. `product = x y`

Syntax errors are analogous to spelling or grammar mistakes in a language like English:

e.g. “Would you some tea?”

does not make sense – it is missing a verb.

Types of Errors – Why (1)

- Common Python syntax errors :
  - leaving out a keyword
  - putting a keyword in the wrong place
  - leaving out a symbol, such as a colon, comma or brackets
  - misspelling a keyword
  - incorrect indentation

Types of Errors – Why (2)

- Logic Error
  - Program passes checking/compiling and runs but produces incorrect results or no results - because of a flaw in the algorithm or implementation of algorithm.
    - e.g. `product = x + y`

Checkpoint

```
def Maximum(x,y):
    z=x
    if (x<y) z=y
    return z
```

What kind of error?

Checkpoint

```
def Maximum(x,y):
    z=x
    if (x>y): z=y
    return z
```

What kind of error?

Checkpoint

```
def Maximum(x,y):
    z=x
    if (x<y): z==y
    return z
```

What kind of error?

Errors: Recipe Analogy

Pancake Recipe:

30 ml olive oil

6 eggs

4 potatoes

1 clove minced garlic

Chop ptatos into small cubes. Vigourously eggs beat. Add chopped onion, garlic and oil to pan and fry till golden. Cut potatoes into thin slices. Pour egg into pan and cook till set.

Errors: Recipe Analogy

Pancake Recipe:

Logic error

30 ml olive oil

6 eggs

4 potatoes

1 clove minced garlic

Compile time errors

Chop **ptatos** into small cubes. **Vigourously eggs beat.** Add **chopped onion, garlic** and oil to **pan** and fry till golden. Cut potatoes into thin slices. Pour egg into pan and cook till set.

Run time error

Testing versus Debugging

Test: run with sample data to uncover errors

Debug: find the cause of a known error

Testing Methods

Programs must be thoroughly tested
for **all possible** input/output values
to make sure the programs behave **correctly**.

Exhaustive testing

Ideal testing strategy :

Run program using all possible inputs

Compare actual outputs to expected outputs

Exhaustive testing

But how do we test for all values of integers?

```
a = eval (input ("Enter a number
between 1 and 100: "))
if a<1 or a>100:
   print ("Error")
)
```

This simple program asks the user for one integer –
how many possible inputs in Python?

Random testing

A subset of values in the input domain is used for testing.

- Important to ensure that values are distributed over input domain
- Can use random number generation.

Equivalence Classes & Boundary Values

Equivalence classes: Group input values into sets of values with similar expected behaviour and choose candidate values

e.g.

- (-50, 150)
- (10, 50, 80)

```
a = eval (input (""))

if a<1 or a>100:
   print ("Error")
```

Boundary value analysis: choose values at, and on either side of, the boundaries of the equivalence classes–

e.g. 0, 1, 2, 99, 100, 101

Equivalence Classes & Boundary Values

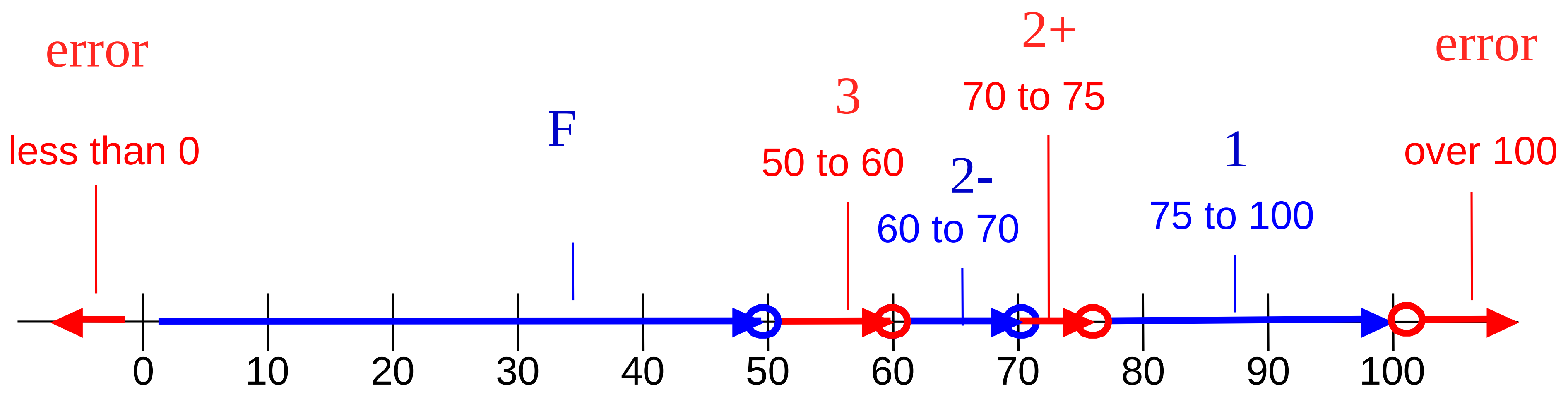
Write a program to classify test scores into ranks:

- 1, 2+, 2-, 3,,F

Equivalence Classes & Boundary Values

Example: Test score program

Equivalence classes:



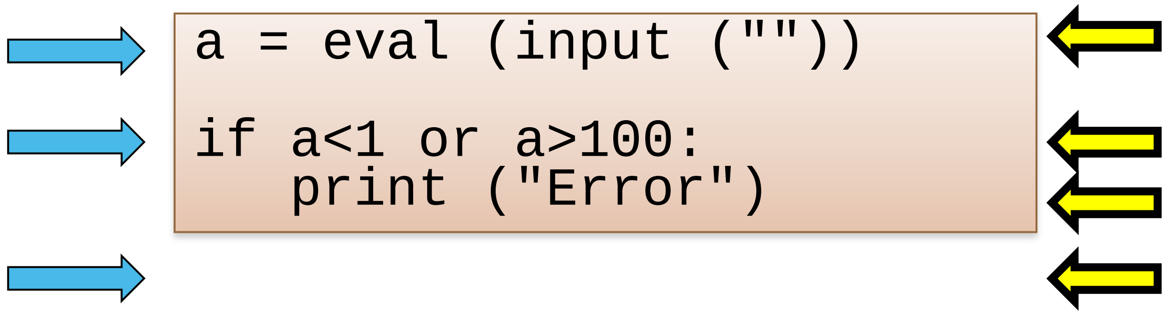
Equivalence Classes & Boundary Values

Equivalence Class	*Sample Value*
Scores greater than 100	150
Scores between 75 and 100	95
Scores between 70 and 75	72
Scores between 60 and 70	65
Scores between 50 and 59	55
Scores between 0 and 50	30
Scores less than 0	-50

Boundary Value	*Just Above Boundary Value*	*Just Below Boundary Value*
100	101	99
75	76	74
70	71	69
60	61	59
50	51	49
0	1	-1

Path Testing

- Create test cases to test **every path** of execution of the program at least **once**.



```
a = eval (input (""))

if a<1 or a>100:
   print ("Error")
```

Path 1: a=35

Path 2: a=-5

UNIVERSITY OF CAPE TOWN

Statement Coverage

- What if we had:

```
if a<25:
   print ("Error in a")
else:
   print ("No error in a")
if b<25:
   print ("Error in b")
else:
   print ("No error in b")
```

- Rather than test all paths, test all statements at least once.
  - e.g., (a,b) = (10, 10), (50, 50)

Glass and Black Boxes

If you can create your test cases based on only the problem specification, it is *black box testing*.

- If you have to examine the code, it is *glass box testing*.

Which categories do these fall into?

- Exhaustive Testing
- Random Testing
- Equivalence classes/boundary values
- Path coverage
- Statement coverage

Checkpoint

```
# CSC1015f: Software Testing sample
# Michelle Kuttel April 2015

# get a number
number= eval(input("Enter a number:\n"))
number= number + 10

# check if it is small or large
if number<20:
   print("small number")
else:
   print("large number")

# check if it is divisible by 5
if number % 5 == 0:
   print(“divisible by 5")
```

Equivalence classes:

Boundary values:

Statement coverage:

Path coverage:

Checkpoint

```
# CSC1015f: Software Testing sample
# Michelle Kuttel April 2015

# get a number
number= eval(input("Enter a number:\n"))
number= number + 10

# check if it is small or large
if number<20:
   print("small number")
else:
   print("large number")

# check if it is divisible by 5
if number % 5 == 0:
   print(“divisible by 5")
```

Equivalence classes:

small multiple of 5: 5
small non-multiples of 5: 3
large multiple of 5: 25
large non-multiples of 5

Boundary values:

Statement coverage:

Path coverage:

Checkpoint

```
# CSC1015f: Software Testing sample
# Michelle Kuttel April 2015

# get a number
number= eval(input("Enter a number:\n"))
number= number + 10

# check if it is small or large
if number<20:
   print("small number")
else:
   print("large number")

# check if it is divisible by 5
if number % 5 == 0:
   print("divisible by 5")
```

Equivalence classes:

small multiple of 5: 5
small non-multiples of 5: 3
large multiple of 5: 25
large non-multiples of 5

Boundary values:

4, 5, 6, 9, 10, 11, 14, 15, 16

Statement coverage:

Path coverage:

Checkpoint

```
# CSC1015f: Software Testing sample
# Michelle Kuttel April 2015

# get a number
number= eval(input("Enter a number:\n"))
number= number + 10

# check if it is small or large
if number<20:
   print("small number")
else:
   print("large number")

# check if it is divisible by 5
if number % 5 == 0:
   print(“divisible by 5")
```

Equivalence classes:

small multiple of 5: 5
small non-multiples of 5: 3
large multiple of 5: 25
large non-multiples of 5

Boundary values:

4, 5, 6, 9, 10, 11, 14, 15,

Statement coverage:

5, 14

Path coverage:

UNIVERSITY OF CAPE TOWN

Checkpoint

```
# CSC1015f: Software Testing sample
# Michelle Kuttel April 2015

# get a number
number= eval(input("Enter a number:\n"))
number= number + 10

# check if it is small or large
if number<20:
   print("small number")
else:
   print("large number")

# check if it is divisible by 5
if number % 5 == 0:
   print(“divisible by 5")
```

Equivalence classes:

small multiple of 5: 5
small non-multiples of 5: 3
large multiple of 5: 25
large non-multiples of 5

Boundary values:

4, 5, 6, 9, 10, 11, 14, 15,

Statement coverage:

5, 14

Path coverage:

5, 7, 13, 20

Checkpoint

```
# CSC1015f: Software Testing sample
# Michelle Kuttel April 2015

# get a number
number= eval(input("Enter a number:\n"))
number= number + 10

# check if it is small or large
if number<20:
   print("small number")
else:
   print("large number")

# check if it is divisible by 5
if number % 5 == 0:
   print(“divisible by 5")
```

Which is best?

Equivalence classes:

small multiple of 5: 5
small non-multiples of 5: 3
large multiple of 5: 25
large non-multiples of 5

Boundary values:

4, 5, 6, 9, 10, 11, 14, 15,

Statement coverage:

5, 14

Path coverage:

5, 7, 13, 20

Opinion

Which of these is the best approach to determine test values?

1. Exhaustive testing of all values
2. Random testing
2. Equivalence classes and boundary values
3. Path testing
4. Statement coverage

Finding Errors

- What if a test case fails? Now what?

- Find the error and remove it, using:
  - Tracing
  - Debugger

Tracing

- Insert temporary statements into code to output values during calculation.
- Very useful when there is no debugger!
- Example:

```
x = y*y*2
z = x+5

print (z)
if z == 13:
   …
```

trace instruction

Debugging

- **Debugging** is the process of finding **errors** or **bugs** in code.
- A **debugger** is a tool for executing an application where the programmer can carefully control execution and inspect data.
- Features include:
  - step through code one instruction at a time
  - viewing variables ("Stack Data" in Wing101)
  - insert and remove breakpoints to pause execution

Wing101 manual has info on debug

Checkpoint

```
def riddle(n):
  if n<=0:
     return 0
  a=1
  b=1
  for i in range(n-2):
     a,b=a+b,a
  return a
```

Equivalence classes:

Boundary values:

Statement coverage:

Path coverage:

Checkpoint

```
def riddle(n):
  if n<=0:
     return 0
  a=1
  b=1
  for i in range(n-2):
     a,b=a+b,a
  return a
```

Equivalence classes:

n <=0 : -5

0<n<3

n>=3

Boundary values:

0,1,2,3,4

Statement coverage:

-5; 8

Path coverage:

-5;2;8

Fibonacci Numbers

In the year 1202 in Italy, the mathematician Leonardo of Pisa, who is better known as Fibonacci, posed this problem:

If a pair of rabbits is placed in an enclosed area,

how many pairs of rabbits will be born there

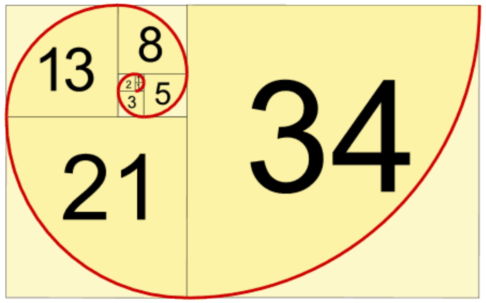
if we assume that every month a pair of rabbits produces another pair,

and that rabbits begin to bear young two months after their birth?

Fibonacci Numbers

Sequence of positive integers such that each number is the sum of the previous two.

- Start with 0 and 1



- Algorithm:
  - `if number=0, result = 0`
  - `if number=1, result = 1`
  - `otherwise, result = fib (number-1) + fib (number-2)`
- Output:
  - 0 1 1 2 3 5 8 13 21 34 55 89 144 233 ... =

- *(fib(n) is the number of pairs of rabbits in month n)*

conditional program execution explained (again?)

Some Python module files are designed to be run directly

- usually referred to as "programs" or "scripts"

Others are designed to be primarily imported and used by other programs

- usually referred to as "libraries"

Sometimes we want both functionalities, a module that can be either imported or run directly (when the "main" function will execute)

conditional program execution explained

A module can determine how it is being run by looking at the value of the `__name__` variable

if it is imported, `__name__` will be the name of the module (e.g. 'math' for math.py)

if it is being run directly, `__name__` will have the value '`__main__`'

So, you can do this to ensure that a module is run when it is invoked directly, but NOT if it is imported

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

Use debugging to show this…

More explanation of conditional program execution

Imagine that you have a Python computer game called SlimeMonsterBlood3

You can either

run this as a game (execute it)

or else import it as a module when creating another game (FairyCastlePrincess)

In this case, you would use the functions in SlimeMonsterBlood3 (castSpell, fightMonster, getAmmunition) as building blocks for your new game

Explaining Conditional program execution

You understand this, right? -

```
if x == True:
  monsterBloodThree()
```

Explaining Conditional program execution

and this, right? -

```
if x == “Run”:
   monsterBloodThree()
```

UNIVERSITY OF CAPE TOWN

Explaining Conditional program execution

Now, `__name__` is just another variable …

```
if __name__ == “Run”:
   monsterBloodThree()
```

Explaining Conditional program execution

And `__main__` is just another literal …

```
if __name__ == ‘__main__’:
  monsterBloodThree()
```

Checkpoint

```
def Sorting(x,y,z):
    if x>y:
        if y>z:
            print(x,y,z)
        elif x>z:
            print(x,z,y)
        else: print(z,x,y)
    elif z>y:
        print(z,y,x)
    elif x>z: print(y,x,z)
    else: print(y,z,x)
```

Equivalence classes:

Boundary values:

Statement coverage:

Path coverage:

Checkpoint

```
def Sorting(x,y,z):
    if x>y:
        if y>z:
            print(x,y,z)
        elif x>z:
            print(x,z,y)
        else: print(z,x,y)
    elif z>y:
        print(z,y,x)
    elif x>z: print(y,x,z)
    else: print(y,z,x)
```

Equivalence classes:

Big, Mid, Small; Big, Small, Mid;

Small, Mid, Big; Small, Big, Mid;

Mid, Small, Big; Mid, Big, Small

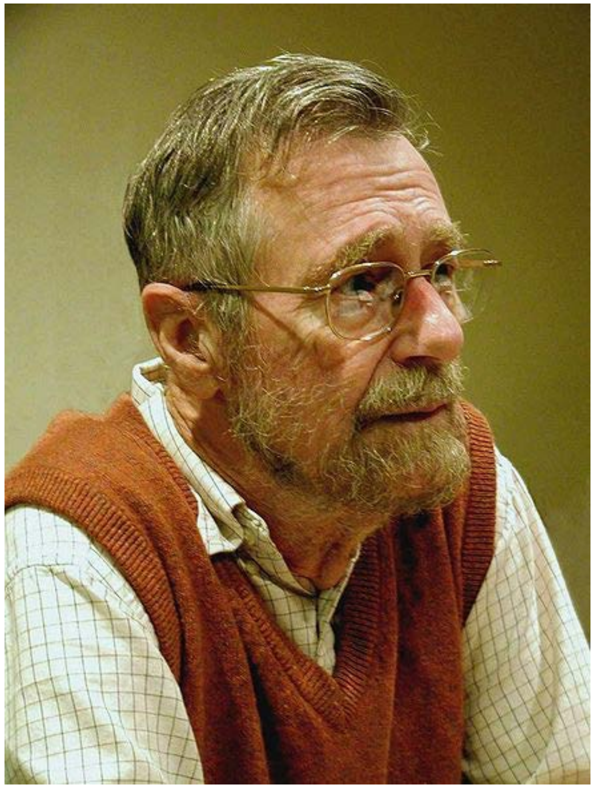
Boundary values:

Statement coverage:

Path coverage:

A quote to end the section

Program testing can, at best, show the presence of errors, but never their absence.



Edsgar Dijkstra

1930-2002

Famous Dutch

Computer Scientist