Toy example for Claus’ paper (for discussion):

Assume we have a class called ArrayCalculator (short: AC) as follows:

* AC has as an instance variable an array of integers with 3 elements (name: *arr*)
* AC has a setter method **set** that assigns values to the instance variable *arr*
* AC has a getter method **get** that displays the elements of the instance variable *arr*
* AC has a method **avg** that returns the average value of the three elements of *arr*
* AC has a method **first** that returns the value of the first element of *arr*
* AC has a method **last** that returns the value of the last element of *arr*

The **get** method is used to read the internal state, i.e., the value of *arr*.

Assume we have the following test cases:

TC1a: AC.set([0,0,0]); assert\_equal(0,AC.first(arr)) – input [0,0,0]

TC1b: AC.set([1,2,3]); assert\_equal(1,AC.first(arr))

TC1c: AC.set([1,4,1]); assert\_equal(1,AC.first(arr))

TC2a: AC.set([0,0,0]); assert\_equal(0,AC.last(arr)) – input [0,0,0]

TC2b: AC.set([1,2,3]); assert\_equal(3,AC.last(arr))

TC2c: AC.set([1,4,1]); assert\_equal(1,AC.last(arr))

TC3a: AC.set([0,0,0]); assert\_equal(0,AC.avg(arr)) – input [0,0,0]

TC3b: AC.set([1,2,3]); assert\_equal(2,AC.avg(arr))

TC3c: AC.set([1,4,1]); assert\_equal(2,AC.avg(arr))

Assume we capture the internal state for each input:

[0,0,0] – AC.get prints 0-0-0

[1,2,3] – AC.get prints 1-2-3

[1,4,1] – AC.get prints 1-4-1

Now let’s inject the following faults:

F1: in **set** always make the last element equal ‘0’

F2: in **get** always print ‘0’ for the last element, i.e., x-x-0

F3: in **first** always change ‘0’ to ‘1’ (thus, the index of the array will be wrong)

F4: in **last** always change ‘2’ to ‘0’ (thus, the index of the array will be wrong)

F5: in **avg** always change ‘0’ to ‘1’ (thus, the index of the array will be wrong, and the counter in the loop will start from the second element in *arr*)

Evaluation of classic mutation testing (i.e., checking for differences in output when input is unchanged):

1. Input is [0,0,0] and **set** has no fault injected, then:

a.1: test of **first** will pass -> mutant not killed

a.2: test of **last** will pass -> mutant not killed

a.3: test of **avg** will pass -> mutant not killed

1. Input is [1,2,3] and **set** has no fault injected, then:

b.1: test of **first** will fail (‘2’ not equal to ‘1’) -> mutant killed

b.2: test of **last** will fail (‘1’ not equal to ‘3’) -> mutant killed

b.3: test of **avg** will fail (‘1.6666…’ not equal to ‘2’) -> mutant killed

1. Input is [1,4,1] and **set** has no fault injected, then:

c.1: test of **first** will fail (‘4’ not equal to ‘1’) -> mutant killed

c.2: test of **last** will pass -> mutant not killed

c.3: test of **avg** will fail (‘1.6666…’ not equal to ‘2’) -> mutant killed

Score for all 9 tests: 5/9 = 45%

in a)

Checking the internal state won’t detect anything – it is always 0-0-0 as in unmutated case

in b)

Checking the internal state won’t detect anything – it is always 1-2-3 as in unmutated case

in c)

Checking the internal state won’t detect anything – it is always 1-4-1 as in unmutated case

Evaluation of classic mutation testing (i.e., checking for differences in output when input is unchanged):

1. Input is [0,0,0] and **set** has fault injected (arr=[0,0,0]), then:

d.1: test of **first** will pass -> mutant not killed

d.2: test of **last** will pass -> mutant not killed

d.3: test of **avg** will pass -> mutant not killed

1. Input is [1,2,3] and **set** has fault injected (arr=[1,2,0]), then:

e.1: test of **first** will fail (‘2’ not equal to ‘1’) -> mutant killed

e.2: test of **last** will fail (‘1’ not equal to ‘3’) -> mutant killed

e.3: test of **avg** will fail (‘1’ not equal to ‘2’) -> mutant killed

1. Input is [1,4,1] and **set** has fault injected (arr=[1,4,0]), then:

f.1: test of **first** will fail (‘4’ not equal to ‘1’) -> mutant killed

f.2: test of **last** will pass -> mutant not killed - NB: we compare arr[0] with ‘1’

f.3: test of **avg** will fail (‘1.6666…’ not equal to ‘2’) -> mutant killed

Score for all 9 tests: 6/9 = 67% - we now also kill f.2 due to changed internal state

in a)

Checking the internal state won’t detect anything – it is always 0-0-0 as in unmutated case

in b)

Checking the internal state will always hint a problem – 1-2-0 differs from 1-2-3

in c)

Checking the internal state will always hint a problem – 1-4-0 differs from 1-4-1

Summary:

1. If none of the state changing method is mutated (no fault injected), checking of internal state will not improve the score
2. If we mutate the state changing method, test all non-state-changing methods and use three different inputs, the score improves from 45% to 67% (but this result is sensitive to the type of fault injection and the choice of inputs)
3. If we mutate the state changing method, test all non-state-changing methods and only use input [0,0,0], the score does not improve (it stays 0%)
4. If we mutate the state changing method, test all non-state-changing methods and only use input [1,2,3], the score does not improve (it stays 100%)
5. If we mutate the state changing method, test all non-state-changing methods and only use input [1,4,1], the score improves from 67% to 100%

Conclusion:

The combination of fault injection to the state-changing method and choice of input data influences the performance:

1. If the input data is chosen such that all mutants in the non-state-changing methods are killed (see case b), then there is no point in checking state changes
2. If the input data is chosen such that mutants in state-changing methods do not trigger a state change (see case d), then there is no point in checking state changes

Goal must be to avoid situations as in case d).

Questions:

1. How often are state-changing methods actually mutated when using mutation operators?
2. How often does it actually happen that wrong choice of input data masks the potential effect of a faulty state-changing method?

**F1: in set always make the last element equal ‘0’**

Evaluation of classic mutation testing (i.e., checking for differences in output when input is unchanged):

**MANUAL example:**

Input is [0,0,0] and **set** has fault injected (arr = [0,0,0]), then:

a.1: test of **first** will pass -> mutant not killed

a.2: test of **last** will pass -> mutant not killed

a.3: test of **avg** will pass -> mutant not killed

Input is [1,2,3] and **set** has fault injected (arr = [1,2,0]), then:

b.1: test of **first** will pass (‘1’ equal to ‘1’) -> mutant not killed

b.2: test of **last** will fail (‘3’ not equal to ‘0’) -> **mutant killed**

b.3: test of **avg** will fail (‘2’ not equal to ‘1’) -> **mutant killed**

Input is [1,4,1] and **set** has fault injected (arr = [1,4,0]), then:

c.1: test of **first** will pass (‘1’ equal to ‘1’) -> mutant not killed

c.2: test of **last** will fail (‘1’ not equal to ‘0’) -> **mutant killed**

c.3: test of **avg** will fail (‘2’ not equal to ‘1.6666…’) -> **mutant killed**

Score for all 9 tests: 4/9 = 44,44...%

Implementation

Test outputs:

---

testOne\_a1 (\_\_main\_\_.TestArryCalc) ... ok

testOne\_b1 (\_\_main\_\_.TestArryCalc) ... ok

testOne\_c1 (\_\_main\_\_.TestArryCalc) ... ok

testTwo\_a2 (\_\_main\_\_.TestArryCalc) ... ok

testTwo\_b2 (\_\_main\_\_.TestArryCalc) ... FAIL

testTwo\_c2 (\_\_main\_\_.TestArryCalc) ... FAIL

testThree\_a3 (\_\_main\_\_.TestArryCalc) ... ok

testThree\_b3 (\_\_main\_\_.TestArryCalc) ... FAIL

testThree\_c3 (\_\_main\_\_.TestArryCalc) ... FAIL

======================================================================

FAIL: testTwo\_b2 (\_\_main\_\_.TestArryCalc)

----------------------------------------------------------------------

Traceback (most recent call last):

File "test\_F1.py", line 61, in testTwo\_b2

self.assertEqual(3, b2.last())

AssertionError: 3 != 0

======================================================================

FAIL: testTwo\_c2 (\_\_main\_\_.TestArryCalc)

----------------------------------------------------------------------

Traceback (most recent call last):

File "test\_F1.py", line 66, in testTwo\_c2

self.assertEqual(1, c2.last())

AssertionError: 1 != 0

======================================================================

FAIL: testThree\_b3 (\_\_main\_\_.TestArryCalc)

----------------------------------------------------------------------

Traceback (most recent call last):

File "test\_F1.py", line 76, in testThree\_b3

self.assertEqual(2, b3.avg())

AssertionError: 2 != 1.0

======================================================================

FAIL: testThree\_c3 (\_\_main\_\_.TestArryCalc)

----------------------------------------------------------------------

Traceback (most recent call last):

File "test\_F1.py", line 81, in testThree\_c3

self.assertEqual(2, c3.avg())

AssertionError: 2 != 1.6666666666666667

----------------------------------------------------------------------

Ran 9 tests in 0.051s

FAILED (failures=4) Score for all 9 tests: 4/9 = 44,44...%

**Checking the internal state and comparing it with the internal state of the first version (code not mutated)**

|  |  |  |
| --- | --- | --- |
|  | **No\_Mutant** | **F1** |
| **testID** | **get** | **get** |
| TC-a1 | [0, 0, 0] | [0, 0, 0] |
| TC-b1 | [1, 2, 3] | [1, 2, 0] |
| T-c1 | [1, 4, 1] | [1, 4, 0] |
| TC-a2 | [0, 0, 0] | [0, 0, 0] |
| TC-b2 | [1, 2, 3] | [1, 2, 0] |
| TC-c2 | [1, 4, 1] | [1, 4, 0] |
| TC-a3 | [0, 0, 0] | [0, 0, 0] |
| TC-b3 | [1, 2, 3] | [1, 2, 0] |
| TC-c3 | [1, 4, 1] | [1, 4, 0] |

**Score for all 9 tests: 6/9 = 67%**

**Extending the “internal State data”**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **State first version (No mutants)** | | | | **State when fault 1 is injected** | | | | |
| **testID** | **get** | **avg** | **first** | **last** | **get** | **avg** | **first** | **last** |
| TC-a1 | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC-b1 | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 0] | 1 | 1 | 0 |
| T-c1 | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 0] | 1.67 | 1 | 0 |
| TC-a2 | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC-b2 | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 0] | 1 | 1 | 0 |
| TC-c2 | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 0] | 1.67 | 1 | 0 |
| TC-a3 | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC-b3 | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 0] | 1 | 1 | 0 |
| TC-c3 | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 0] | 1.67 | 1 | 0 |

**Score for all 9 tests: 6/9 = 67%**

**F2: in get always print ‘0’ for the last element, i.e., x-x-0**

Evaluation of classic mutation testing (i.e., checking for differences in output when input is unchanged):

**MANUAL example:**

Input is [0,0,0] and **get** has fault injected (will return arr = [0,0,0]), then:

a.1: test of **first** will pass -> mutant not killed

a.2: test of **last** will pass -> mutant not killed

a.3: test of **avg** will pass -> mutant not killed

Input is [1,2,3] and **get** has fault injected (will return arr = [1,2,0]), then:

b.1: test of **first** will pass (‘1’ equal to ‘1’) -> mutant not killed

b.2: test of **last** will fail (‘3’ not equal to ‘0’) -> **mutant killed**

b.3: test of **avg** will pass (‘2’ not equal to ‘2’) -> mutant not killed

Input is [1,4,1] and **get** has fault injected (will return arr = [1,4,0]), then:

c.1: test of **first** will pass (‘1’ equal to ‘1’) -> mutant not killed

c.2: test of **last** will fail (‘1’ not equal to ‘0’) -> **mutant killed**

c.3: test of **avg** will fail (‘2’ not equal to ‘2’) -> mutant not killed

**Implementation**

**Tests output:**

**---**

**testOne\_a1 (\_\_main\_\_.TestArryCalc) ... ok**

**testOne\_b1 (\_\_main\_\_.TestArryCalc) ... ok**

**testOne\_c1 (\_\_main\_\_.TestArryCalc) ... ok**

**testTwo\_a2 (\_\_main\_\_.TestArryCalc) ... ok**

**testTwo\_b2 (\_\_main\_\_.TestArryCalc) ... FAIL**

**testTwo\_c2 (\_\_main\_\_.TestArryCalc) ... FAIL**

**testThree\_a3 (\_\_main\_\_.TestArryCalc) ... ok**

**testThree\_b3 (\_\_main\_\_.TestArryCalc) ... FAIL**

**testThree\_c3 (\_\_main\_\_.TestArryCalc) ... FAIL**

**=====================================================================**

**FAIL: testTwo\_b2 (\_\_main\_\_.TestArryCalc)**

**----------------------------------------------------------------------**

**Traceback (most recent call last):**

**File "test\_F2.py", line 61, in testTwo\_b2**

**self.assertEqual(3, b2.last())**

**AssertionError: 3 != 0**

**======================================================================**

**FAIL: testTwo\_c2 (\_\_main\_\_.TestArryCalc)**

**----------------------------------------------------------------------**

**Traceback (most recent call last):**

**File "test\_F2.py", line 66, in testTwo\_c2**

**self.assertEqual(1, c2.last())**

**AssertionError: 1 != 0**

**======================================================================**

**FAIL: testThree\_b3 (\_\_main\_\_.TestArryCalc)**

**----------------------------------------------------------------------**

**Traceback (most recent call last):**

**File "test\_F2.py", line 76, in testThree\_b3**

**self.assertEqual(2, b3.avg())**

**AssertionError: 2 != 1.0**

**======================================================================**

**FAIL: testThree\_c3 (\_\_main\_\_.TestArryCalc)**

**----------------------------------------------------------------------**

**Traceback (most recent call last):**

**File "test\_F2.py", line 81, in testThree\_c3**

**self.assertEqual(2, c3.avg())**

**AssertionError: 2 != 1.6666666666666667**

**---------------------------------------------------------------------**

**Ran 9 tests in 0.049s**

**FAILED (failures=4) Score for all 9 tests: 4/9 = 44,44...%**

**Checking the internal state and comparing it with the internal state of the first version (code not mutated)**

|  |  |  |
| --- | --- | --- |
|  | **No\_Mutant** | **F2** |
| **testID** | **get** | **get** |
| TC-a1 | [0, 0, 0] | [0, 0, 0] |
| TC-b1 | [1, 2, 3] | [1, 2, 0] |
| T-c1 | [1, 4, 1] | [1, 4, 0] |
| TC-a2 | [0, 0, 0] | [0, 0, 0] |
| TC-b2 | [1, 2, 3] | [1, 2, 0] |
| TC-c2 | [1, 4, 1] | [1, 4, 0] |
| TC-a3 | [0, 0, 0] | [0, 0, 0] |
| TC-b3 | [1, 2, 3] | [1, 2, 0] |
| TC-c3 | [1, 4, 1] | [1, 4, 0] |

**Score for all 9 tests: 6/9 = 67%**

**Extending the “internal State data”**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **State first version (No mutants)** | | | | **State when fault 2 is injected** | | | | |
| **testID** | **get** | **avg** | **first** | **last** | **get** | **avg** | **first** | **last** |
| TC-a1 | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC-b1 | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 0] | 1 | 1 | 0 |
| T-c1 | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 0] | 1.67 | 1 | 0 |
| TC-a2 | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC-b2 | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 0] | 1 | 1 | 0 |
| TC-c2 | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 0] | 1.67 | 1 | 0 |
| TC-a3 | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC-b3 | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 0] | 1 | 1 | 0 |
| TC-c3 | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 0] | 1.67 | 1 | 0 |

**Score for all 9 tests: 6/9 = 67%**

**F3: in first always change ‘0’ to ‘1’ (thus, the index of the array will be wrong)**

**MANUAL example:**

Input is [0,0,0] and **first** has fault injected (will return 0), then:

a.1: test of **first** will pass -> mutant not killed

a.2: test of **last** will pass -> mutant not killed

a.3: test of **avg** will pass -> mutant not killed

Input is [1,2,3] and **first** has fault injected (will return 2), then:

b.1: test of **first** will fail (‘1’ not equal to ‘2’) -> **mutant killed**

b.2: test of **last** will pass (‘3’ equal to ‘3’) -> mutant not killed

b.3: test of **avg** will pass (‘2’ not equal to ‘2’) -> mutant not killed

Input is [1,4,1] and **first** has fault injected (will return 4), then:

c.1: test of **first** will fail (‘1’ equal to ‘4’) -> **mutant killed**

c.2: test of **last** will pass (‘1’ equal to ‘1’) -> mutant not killed

c.3: test of **avg** will pass (‘2’ equal to ‘2’) -> mutant not killed

**Implementation**

**Tests output:**

**---**

**testOne\_a1 (\_\_main\_\_.TestArryCalc) ... ok**

**testOne\_b1 (\_\_main\_\_.TestArryCalc) ... FAIL**

**testOne\_c1 (\_\_main\_\_.TestArryCalc) ... FAIL**

**testTwo\_a2 (\_\_main\_\_.TestArryCalc) ... ok**

**testTwo\_b2 (\_\_main\_\_.TestArryCalc) ... ok**

**testTwo\_c2 (\_\_main\_\_.TestArryCalc) ... ok**

**testThree\_a3 (\_\_main\_\_.TestArryCalc) ... ok**

**testThree\_b3 (\_\_main\_\_.TestArryCalc) ... ok**

**testThree\_c3 (\_\_main\_\_.TestArryCalc) ... ok**

**======================================================================**

**FAIL: testOne\_b1 (\_\_main\_\_.TestArryCalc)**

**----------------------------------------------------------------------**

**Traceback (most recent call last):**

**File "test\_F3.py", line 46, in testOne\_b1**

**self.assertEqual(1, b1.first())**

**AssertionError: 1 != 2**

**======================================================================**

**FAIL: testOne\_c1 (\_\_main\_\_.TestArryCalc)**

**----------------------------------------------------------------------**

**Traceback (most recent call last):**

**File "test\_F3.py", line 51, in testOne\_c1**

**self.assertEqual(1, c1.first())**

**AssertionError: 1 != 4**

**----------------------------------------------------------------------**

**Ran 9 tests in 0.063s**

**FAILED (failures=2) Score for all 9 tests: 2/9 = 22,22...%**

**Checking the internal state and comparing it with the internal state of the first version (code not mutated)**

|  |  |  |
| --- | --- | --- |
|  | **No\_Mutant** | **F3** |
| **testID** | **get** | **get** |
| TC1a | [0, 0, 0] | [0, 0, 0] |
| TC1b | [1, 2, 3] | [1, 2, 3] |
| TC1c | [1, 4, 1] | [1, 4, 1] |
| TC2a | [0, 0, 0] | [0, 0, 0] |
| TC2b | [1, 2, 3] | [1, 2, 3] |
| TC2c | [1, 4, 1] | [1, 4, 1] |
| TC3a | [0, 0, 0] | [0, 0, 0] |
| TC3b | [1, 2, 3] | [1, 2, 3] |
| TC3c | [1, 4, 1] | [1, 4, 1] |

**Score for all 9 tests: 6/9 = 67%**

**Extending the “internal State data”**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **State first version (No mutants)** | | | |  | **State when fault 3 is injected** | | | |
| **testID** | **get** | **avg** | **first** | **last** |  | **get** | **avg** | **first** | **last** |
| TC1a | [0, 0, 0] | 0 | 0 | 0 |  | [0, 0, 0] | 0 | 0 | 0 |
| TC1b | [1, 2, 3] | 2 | 1 | 3 |  | [1, 2, 3] | 2 | 2 | 3 |
| TC1c | [1, 4, 1] | 2 | 1 | 1 |  | [1, 4, 1] | 2 | 4 | 1 |
| TC2a | [0, 0, 0] | 0 | 0 | 0 |  | [0, 0, 0] | 0 | 0 | 0 |
| TC2b | [1, 2, 3] | 2 | 1 | 3 |  | [1, 2, 3] | 2 | 2 | 3 |
| TC2c | [1, 4, 1] | 2 | 1 | 1 |  | [1, 4, 1] | 2 | 4 | 1 |
| TC3a | [0, 0, 0] | 0 | 0 | 0 |  | [0, 0, 0] | 0 | 0 | 0 |
| TC3b | [1, 2, 3] | 2 | 1 | 3 |  | [1, 2, 3] | 2 | 2 | 3 |
| TC3c | [1, 4, 1] | 2 | 1 | 1 |  | [1, 4, 1] | 2 | 4 | 1 |

**Score for all 9 tests: 6/9 = 67%**

**F4: in last always change ‘2’ to ‘0’ (thus, the index of the array will be wrong)**

**MANUAL example:**

Input is [0,0,0] and **last** has fault injected (will return 0), then:

a.1: test of **first** will pass -> mutant not killed

a.2: test of **last** will pass -> mutant not killed

a.3: test of **avg** will pass -> mutant not killed

Input is [1,2,3] and **last** has fault injected (will return 1), then:

b.1: test of **first** will pass (‘1’ equal to ‘1’) -> mutant not killed

b.2: test of **last** will fail (‘3’ not equal to ‘1’) -> **mutant killed**

b.3: test of **avg** will pass (‘2’ equal to ‘2’) -> mutant not killed

Input is [1,4,1] and **last** has fault injected (will return 1), then:

c.1: test of **first** will pass (‘1’ equal to ‘1’) -> mutant not killed

c.2: test of **last** will pass (‘1’ equal to ‘1’) -> mutant not killed

c.3: test of **avg** will pass (‘2’ not equal to ‘2’) -> mutant not killed

**Implementation**

**Tests output:**

**---**

**testOne\_a1 (\_\_main\_\_.TestArryCalc) ... ok**

**testOne\_b1 (\_\_main\_\_.TestArryCalc) ... ok**

**testOne\_c1 (\_\_main\_\_.TestArryCalc) ... ok**

**testTwo\_a2 (\_\_main\_\_.TestArryCalc) ... ok**

**testTwo\_b2 (\_\_main\_\_.TestArryCalc) ... FAIL**

**testTwo\_c2 (\_\_main\_\_.TestArryCalc) ... ok**

**testThree\_a3 (\_\_main\_\_.TestArryCalc) ... ok**

**testThree\_b3 (\_\_main\_\_.TestArryCalc) ... ok**

**testThree\_c3 (\_\_main\_\_.TestArryCalc) ... ok**

**======================================================================**

**FAIL: testTwo\_b2 (\_\_main\_\_.TestArryCalc)**

**----------------------------------------------------------------------**

**Traceback (most recent call last):**

**File "test\_F4.py", line 61, in testTwo\_b2**

**self.assertEqual(3, b2.last())**

**AssertionError: 3 != 1**

**----------------------------------------------------------------------**

**Ran 9 tests in 0.050s**

**FAILED (failures=1) Score for all 9 tests: 1/9 = 11,11...%**

**Checking the internal state and comparing it with the internal state of the first version (code not mutated)**

|  |  |  |
| --- | --- | --- |
|  | **No\_Mutant** | **F4** |
| **testID** | **get** | **get** |
| TC1a | [0, 0, 0] | [0, 0, 0] |
| TC1b | [1, 2, 3] | [1, 2, 3] |
| TC1c | [1, 4, 1] | [1, 4, 1] |
| TC2a | [0, 0, 0] | [0, 0, 0] |
| TC2b | [1, 2, 3] | [1, 2, 3] |
| TC2c | [1, 4, 1] | [1, 4, 1] |
| TC3a | [0, 0, 0] | [0, 0, 0] |
| TC3b | [1, 2, 3] | [1, 2, 3] |
| TC3c | [1, 4, 1] | [1, 4, 1] |

**Score for all 9 tests = 0**

**Extending the “internal State data”**

**Checking the internal state and comparing it with the internal state of the first version (code not mutated)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **State first version (No mutants)** | | | | **State when fault 4 is injected** | | | |
| **testID** | **get** | **avg** | **first** | **last** | **get** | **avg** | **first** | **last** |
| TC1a | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC1b | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 3] | 2 | 1 | 1 |
| TC1c | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 1] | 2 | 1 | 1 |
| TC2a | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC2b | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 3] | 2 | 1 | 1 |
| TC2c | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 1] | 2 | 1 | 1 |
| TC3a | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC3b | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 3] | 2 | 1 | 1 |
| TC3c | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 1] | 2 | 1 | 1 |

**Score for all 9 tests: 3/9 = 33,33…%**

**F5: in avg always change ‘0’ to ‘1’ (thus, the index of the array will be wrong and the counter in the loop will start from the second element in *arr*)**

**MANUAL example:**

Input is [0,0,0] and **avg** has fault injected (will return 0), then:

a.1: test of **first** will pass -> mutant not killed

a.2: test of **last** will pass -> mutant not killed

a.3: test of **avg** will pass -> mutant not killed

Input is [1,2,3] and **avg** has fault injected (will return 1,66), then:

b.1: test of **first** will pass (‘1’ equal to ‘1’) -> mutant not killed

b.2: test of **last** will pass (‘3’ equal to ‘3’) -> mutant not killed

b.3: test of **avg** will fail (‘2’ Not equal to ‘1,66’) -> **mutant killed**

Input is [1,4,1] and **last** has fault injected (will return 1), then:

c.1: test of **first** will pass (‘1’ equal to ‘1’) -> mutant not killed

c.2: test of **last** will pass (‘1’ equal to ‘1’) -> mutant not killed

c.3: test of **avg** will pass (‘2’ not equal to ‘1,66’’) -> **mutant killed**

**Implementation**

**Tests output:**

**---**

**testOne\_a1 (\_\_main\_\_.TestArryCalc) ... ok**

**testOne\_b1 (\_\_main\_\_.TestArryCalc) ... ok**

**testOne\_c1 (\_\_main\_\_.TestArryCalc) ... ok**

**testTwo\_a2 (\_\_main\_\_.TestArryCalc) ... ok**

**testTwo\_b2 (\_\_main\_\_.TestArryCalc) ... ok**

**testTwo\_c2 (\_\_main\_\_.TestArryCalc) ... ok**

**testThree\_a3 (\_\_main\_\_.TestArryCalc) ... ok**

**testThree\_b3 (\_\_main\_\_.TestArryCalc) ... FAIL**

**testThree\_c3 (\_\_main\_\_.TestArryCalc) ... FAIL**

**======================================================================**

**FAIL: testThree\_b3 (\_\_main\_\_.TestArryCalc)**

**----------------------------------------------------------------------**

**Traceback (most recent call last):**

**File "test\_F5.py", line 76, in testThree\_b3**

**self.assertEqual(2, b3.avg())**

**AssertionError: 2 != 1.6666666666666667**

**======================================================================**

**FAIL: testThree\_c3 (\_\_main\_\_.TestArryCalc)**

**----------------------------------------------------------------------**

**Traceback (most recent call last):**

**File "test\_F5.py", line 81, in testThree\_c3**

**self.assertEqual(2, c3.avg())**

**AssertionError: 2 != 1.6666666666666667**

**----------------------------------------------------------------------**

**Ran 9 tests in 0.085s**

**FAILED (failures=2) -> Score for all 9 tests: 2/9 = 22,22...%**

**Checking the internal state and comparing it with the internal state of the first version (code not mutated)**

|  |  |  |
| --- | --- | --- |
|  | **No\_Mutant** | **F5** |
| **testID** | **get** | **get** |
| TC1a | [0, 0, 0] | [0, 0, 0] |
| TC1b | [1, 2, 3] | [1, 2, 3] |
| TC1c | [1, 4, 1] | [1, 4, 1] |
| TC2a | [0, 0, 0] | [0, 0, 0] |
| TC2b | [1, 2, 3] | [1, 2, 3] |
| TC2c | [1, 4, 1] | [1, 4, 1] |
| TC3a | [0, 0, 0] | [0, 0, 0] |
| TC3b | [1, 2, 3] | [1, 2, 3] |
| TC3c | [1, 4, 1] | [1, 4, 1] |

**Score for all 9 tests: 0**

**Extending the “internal State data.”**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **State first version (No mutants)** | | | | **State when fault 5 is injected** | | | | |
| **testID** | **get** | **avg** | **first** | **last** | **get** | **avg** | **first** | **last** |
| TC1a | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC1b | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 3] | 1.67 | 1 | 3 |
| TC1c | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 1] | 1.67 | 1 | 1 |
| TC2a | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC2b | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 3] | 1.67 | 1 | 3 |
| TC2c | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 1] | 1.67 | 1 | 1 |
| TC3a | [0, 0, 0] | 0 | 0 | 0 | [0, 0, 0] | 0 | 0 | 0 |
| TC3b | [1, 2, 3] | 2 | 1 | 3 | [1, 2, 3] | 1.67 | 1 | 3 |
| TC3c | [1, 4, 1] | 2 | 1 | 1 | [1, 4, 1] | 1.67 | 1 | 1 |

**Score for all 9 tests: 6/9 = 67%**

Using only **Get** as internal state data

|  |  |  |
| --- | --- | --- |
|  | **Mutation Testing** | **By checking the state** |
| **F1** | 44.44% | 66.67% |
| **F2** | 44.44% | 66.67% |
| **F3** | 22.22% | 0.00% |
| **F4** | 11.11% | 0.00% |
| **F5** | 22.22% | 0.00% |

**Using the state data of the object**. ***“Object state data is the set of the values of all defined attributes of an object at a certain point of time.”***

|  |  |  |
| --- | --- | --- |
|  | **Mutation Testing** | **State Data** |
| **F1** | 44.44% | 66.67% |
| **F2** | 44.44% | 66.67% |
| **F3** | 22.22% | 66.67% |
| **F4** | 11.11% | 33.33% |
| **F5** | 22.22% | 66.67% |