**Assignment 1**

**Neerja Narayanappa**

**1001575625**

**1. How are faults and failures related to testing and debugging? (5 points)**

**Solution:**

A Static defect in the software is called a Fault and External incorrect behavior with respect to the requirements or other description of the expected behavior is called a Failure.

Given reference to the Beizer’s level that we are working it is believed that testing is a process of trying to cause failure and showcase that they are occurring at an acceptable rate. But on the contrary debugging is a diagnostic process when there is a failure an effort is made to find the related fault.

**2. What is verification and validation?**

**Solution:**

**Validation:**The process of evaluating software at the end of software development to ensure compliance with intended usage.

* *Are we building the right product?*

**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 product right?*

**3. What is static analysis and dynamic testing?**

**Solution:**

**Static Analysis**:Testing without executing the program. This includes software inspections and some forms of analysis. Usually static testing is referred to as verification activities.

**Dynamic Testing:**Testing by executing the program with real inputs. Usually “testing” is referred to as dynamic testing.

**4. For what do testers use automation? What are the limitations of automation?**

**Solution:**

Testers use automation for the following reasons:

1. Eliminating excise tasks eliminates drudgery, thereby making the test engineers job more satisfying.

2. Automation can help in many areas, most often to relieve the tester from repetitive, mechanical tasks.

3. Automation frees up time to focus on the fun and challenging parts of testing, namely the revenue tasks.

4. Checking of testing criteria can be automated through instrumentation, which allows a higher level of testing to be performed.

5. Automation can help eliminate errors of omission, such as failing to update all the relevant files with the new set of expected results

6. Automation eliminates some of the variance in test quality caused by differences in individual’s abilities.

**Limitation of Automation:**

1. Automation will always run into undecidable problems, such as infeasible paths, test case generation, internal variables, etc.

2. Automation cannot help validating output or make creative decisions.

**5. Below are four faulty programs. Each includes a test case that results in failure. Answer the following questions about each program.**

**Solution:**

**Program1:**

**(a) Identify the fault.**

The for-loop should include the 0 index:

**for (int i=x.length-1; i >= 0; i--)**

**(b) If possible, identify a test case that does not execute the fault.**

Before the loop test is evaluated a null value for x will result in a NullPointerException, hence there is no execution of the fault.

Test:  x = null, y = 3

Expected Output: NullPointerException

Actual Output: NullPointerException

**(c) If possible, identify a test case that executes the fault, but does not result in an error state**

There is no error for any input where y appears in the second or later position. Also, there is no error, if x is empty.

Test:  x = [2, 3, 5]; y = 3;

Expected Output: 1

Actual Output: 1

**(d) If possible identify a test case that results in an error, but not a failure.**

The missing path is an error for an input where y is not in x, but there is no failure.

Test:  x = [2, 3, 5]; y = 7;

Expected Output: -1

Actual Output: -1

**(e) For the given test case, identify the first error state. Be sure to describe the complete state**

The error state is that the Program counter is outside the loop. In a correct program, the Program Counter should be at the if-test, with index i==0.

Test:  x = [2, 3, 5]; y = 2;

Expected Output: 0

Actual Output: -1

First Error State:

x = [2, 3, 5]

y = 2;

i = 0

Program counter = just before return -1;;

**(f) Fix the fault and verify that the given test now produces the expected output.**

The for-loop should include the 0 index:

for (int i=x.length-1; i >**=** 0; i--)

Test:  x = [2, 3, 5]; y = 2;

Expected Output: 0

Actual Output: 0

**Program2:**

**(a) Identify the fault.**

The test in the conditional should be:

**if (x[i] > 0) {**

**(b) If possible, identify a test case that does not execute the fault.**

All inputs result in the fault being executed.Hence x must be either null or empty. We give the empty case here.

Test: x[]

Expected Output: 0

Actual Output: 0

**(c) If possible, identify a test case that executes the fault, but does not result in an error state**

Any nonempty x without a 0 entry would work.

Test:  x = [5, 6, 7]

Expected Output: 3

Actual Output: 3

**(d) If possible identify a test case that results in an error, but not a failure.**

-Every input that results in error also results in failure. The reason is that error states are not repairable by subsequent processing.

-If there is a 0 in x, all subsequent states will be error states no matter what else is in x.

**(e) For the given test case, identify the first error state. Be sure to describe the complete state**

* Test:  x = [-4, 2, 0, 2]
* Expected Output:  2
* Actual Output: 3
* First Error State:
* x = [-4, 2, 0, 2]
* i = 2;
* count = 1;
* PC = immediately before the count++ statement.

**(f) Fix the fault and verify that the given test now produces the expected output.**

The test in the conditional should be:  if (x[i] > 0) {

Test:  x = [-4, 2, 0, 2];

Expected Output: 2

Actual Output: 2

**Program3:**

**(a) Identify the fault.**

The for-loop should search from high to low:

**for (int i=x.length-1; i >= 0; i--) {**

**(b) If possible, identify a test case that does not execute the fault.**

* All inputs execute the fault - even the null input.

Test: x = [0,1,0]

Expected Output: 2

**(c) If possible, identify a test case that executes the fault, but does not result in an error state**

There is no error if the loop is not executed at all. If the loop is executed only once, high-to-low and low-to-high evaluation are the same. Hence there is no error for length 0 or length 1 inputs.

Test:  x = [5]

Expected Output: -1

Actual Output: -1

**(d) If possible identify a test case that results in an error, but not a failure.**

There would be an error anytime the loop is executed more than once, since the values of index i ascend instead of descend.

Test:  x = [1, 0, 3]

Expected Output: 1

Actual Output: 1

**(e) For the given test case, identify the first error state. Be sure to describe the complete state**

The first error state is when index i has the value 0 when it should have a value at the end of the array, namely x.length-1.

Hence, the first error state is encountered immediately after the assignment to i in the for-statement if there is more than one value in x.

Test:  x = [0, 1, 0]

Expected Output: 2

Actual Output: 0

First Error State:

x = [0, 1, 0]

i=0

PC = just after i= 0;

**(f) Fix the fault and verify that the given test now produces the expected output.**

The for-loop should search from high to low:  for (int i=x.length-1; i >= 0; i--) {

Test:  x = [0, 1, 0]

Expected Output: 2

Actual Output: 2

**Program4:**

**(a) Identify the fault.**

The if-test needs to take account of negative values (positive odd numbers are taken care of by the second test):

**if (x[i]%2 == -1 || x[i] > 0)**

**(b) If possible, identify a test case that does not execute the fault.**

x must be either null or empty. All other inputs result in the fault being executed. We give the empty case here.

Test: x = []

Expected Output: 0

Actual Output: 0

**(c) If possible, identify a test case that executes the fault, but does not result in an error state**

Any nonempty x with only non-negative elements works, this is due to the first part of the compound if-test is not necessary unless the value is negative.

Test: x = [1, 2, 3]

Expected Output: 3

Actual Output: 3

**(d) If possible identify a test case that results in an error, but not a failure.**

For this particular program, every input that results in error also results in failure. The reason is that error states are not repairable by subsequent processing. If there is a negative value in x, all subsequent states will be error states no matter what else is in x.

**(e) For the given test case, identify the first error state. Be sure to describe the complete state**

* Test:  x = [-3, -2, 0, 1, 4]
* Expected Output: 3
* Actual Output: 2
* First Error State:
* x = [-3, -2, 0, 1, 4]
* i = 0;
* count = 0;
* PC = at end of if statement, instead of just before count++

**(f) Fix the fault and verify that the given test now produces the expected output.**

The if-test needs to take account of negative values if (x[i]%2 == -1 || x[i] > 0)

Test: x = [-3, -2, 0, 1, 4]

Expected output = 3

Actual Output = 3

===========================END==========================